The GOALS Approach: Business and Software Modeling Traceability by means of Human-Computer Interaction

DOCTORAL THESIS

Pedro Dionísio Valente

DOCTORATE IN INFORMATICS ENGINEERING

SPECIALTY: SOFTWARE ENGINEERING

UNIVERSIDADE da MADEIRA

A Nossa Universidade

www.uma.pt

July | 2017

The GOALS Approach: Business and Software Modeling Traceability by means of Human-Computer Interaction

Enterprise Modeling Language and Method

PhD Thesis

Pedro Valente

Supervisors

Nuno Nunes

Marco Winckler

Universidade da Madeira

Acknowledgements

I wish to thank my Professor Nuno Nunes, who believed in me from the beginning of the project, and to Professor Marco Winckler who further supported me in believing in myself.

I wish to thank the UMa Rector, Professor José Carmo and his Rectory, who gave me personal support to go to France in order to work in my thesis, as requested by me and my supervisors.

I wish to thank my Mother and Father, who gave all the support that I needed when I needed.

But mostly, I wish to thank Micaela, and Paulinho also, who were with me from the first moment until the very last one. And also to my family of friends.

Abstract

The management of an enterprise relies on the continuous organization and development of its business and software systems. A process that requires merging the ideas of the enterprise' systems managers, targeting the specification of business requirements and the conception and implementation of a supporting information system. This process finds obstacles in the identification and communication of requirements, and also in their transformation in software artefacts, leading to difficulties or loss of traceability between business and software models. Existing methods, languages and techniques are still not sufficiently standardized to ensure that when a business improvement is introduced, the supportive software solution will be implemented within budget and time. Methods are still too closed to the concepts of their original scientific domains, conceiving solutions which are not representative of the business and software conceptual relation and of the complexity concealed in an improvement effort, namely concerning usability and user experience. Moreover, the lack of a common modeling language and method for the conception of holistic and traceable software solutions, also refrains the performance of the enterprise development process. The GOALS Approach presents a solution to surpass these barriers by means of the specification of an enterprise modeling language that relates the business and software conceptual structures using a shared set of concepts, a notation, process, method and techniques, that allow the design of the software as a result of the business organization, ensuring traceability by means of the permanent representation of the business structure in the software structure.

Keywords: Enterprise Architecture, Enterprise Engineering, Human-Computer Interaction, Software Engineering, Software Architecture, Language, Traceability.

Contents

List of Figures

List of Tables

List of Acronyms

AS – Aggregation Space
BP – Business Process
BPI – Business Process Improvement
BR – Business Rule
DBSR – Database System Responsibility
DE – Data Entity
EDP – Enterprise Development Process
IC – Interaction Component
IO – Interaction Object
IX – Interaction Space
ROI – Return of the Investment
SDP – Software Development Process
UISR – User Interface System Responsibility
UT – User Task
UX – User Experience
ZF – Zachman Framework

Formatting

concept - italic – business and/or software modeling concept.
Model - capitalized - business and/or software model.
Technique - italic capitalized - business and/or software modeling technique.
METHOD – all capitalized - business and/or software modeling method or language.
Goals Concept (GC) - capitalized (acronym) - GOALS modeling language concept, or only acronym.

1. Introduction

Traceability of business and software modeling is a requisite concerning business and Information Technology (IT) alignment, and may be perceived as the cornerstone of enterprise development performance. Traceability is the establishment of a relation between business and software concepts that allows the recognition of which components of the business are supported by which components of the Information System, and what may be the impact of changing business *requirements* over the software structure. Complementarily, by means of traceability, it is possible to acknowledge which aspects of the business are still not supported by a software system (and which are), so that new improvements may be identified. These improvements may be introduced at an acceptable rate, especially if the translation of business and software models is avoided [de Vries et al., 2017; Ouyang et al., 2009]. The translation process overhead is in fact what justifies the introduction of a common language of shared concepts that may be used by business and software modelers to carry on the daily Enterprise Development Process (EDP), the process that coordinates the business and software modeling and implementation tasks.

The introduction of a new language in this context requires that the coordination of the ideas of business and software modelers concerning the enterprise are concealed, which may not be straightforward, as the perspectives of both, observing the same enterprise problem, are distinct. For business modelers, the problem is mostly related to the enactment and enhancement of *business processes*, or Business Process Improvements (BPI), where people and software represent solutions; whereas, for software modelers, those solutions are their analysis, design and implementation problem. To surpass this barrier, the concepts used in the dialogue must be properly understood, given the complexity and accuracy needed in the process of conceiving the enterprise solution. Nevertheless, the establishment of this language is possible, since on one hand, the business concepts with a bigger impact on software development may be recognized by software modelers (e.g. the *business process*, *actor* and *regulation* concepts), and on the other hand, business modelers have general knowledge over the more fundamental software concepts (e.g. User Interface, programmed code and Database), as basic premises of a business and software modeling language. Therefore, if the language is close to this high-level of abstraction, the communication may be viable (effective) and fluent (efficient), instead of being an important obstacle [Ferreira & Wazlawick, 2011].

Until now, methods that propose solutions to surpass the conceptual barrier between business and software modeling, do not represent the full extension of the problem and related solution, or do not apply techniques that allow the methodical elaboration of a holistic software architecture for a given BPI, while maintaining traceability at the same time. We find that the main reason is that methods maintain the used concepts closed to the scientific domains where they are originated from, with few or sometimes unstructured extensions, and as a result, not all the complexity that needs to be represented and managed, is addressed in a single conceptual and traceable structure. Existing methods relate business and software artefacts, but typically: (i) do not represent all the relevant business and software development artefacts; (ii) do not provide the needed level of detail; or, (iii) leave room for uncertainty in the specification of the relation between the business and software, which is a critical modeling when targeting traceability. As a result, both: business and software architectural control [Perry & Wolf, 1992]; and, the application of agile development solutions [Beck et al., 2001], are difficult to attain.

This thesis describes "The Goals Approach", which aims at serving as a common language to support EDP. By conveying the perspectives which are usually involved in in-house software development and business management, we elaborated a method called "The Goals Approach", originated from the "Goals Software Construction Process" [Valente, 2009], where "Goals", an acronym, was ideated as "**G**oal-**O**riented **A**pproach (**L**ed) **S**oftware", expressing the target of letting the software development being "led" by the *goals* of the *business processes*, now abbreviated to Goals Approach, and referred as "GOALS" throughout the thesis. Our work focuses in-house software development, for the reason that many enterprises should be aware that they can develop their own software, and because it is where it is originated from.

GOALS is based on a single language, modeling: the customer service and *business processes* of the enterprise; and: the User Interface, the Business Logic, and the Database of the Information System, given a Model-View-Controller (MVC) pattern [Zukowski, 2013]. It produces a software architecture that includes the business architecture as its "back-bone", ensuring traceability of business and software. By providing: a single structure based on a single architectural language; and the application of each technique; in a single method, we believe that we are creating conditions to enable the fluent dialogue of the (usually) multi-disciplinary teams that carry on the daily enterprise development practice (or EDP).

Our method provides support for the following development roles of practitioners:

- *Business Owner*, by providing a holistic view of the business and software, and an EDP that guides the modeling of both structures using a single method and language;
- *Service Designer*, by providing a tool to elaborate the customer interaction with the business, identify applicable *regulations*, the used and produced *resources* and systems;
- *Business Process Manager*, by proving a tool to design *business processes* with enhanced detail concerning their *tasks*, the used spaces, the *regulations* and *data*, using modeling concepts which are understood in the software domain.

And complementarily, by providing a tool for software modelers, namely:

- *Software Architect*, by providing a holistic view of the Information System context, a process, a method, techniques and a modeling language that facilitates the elaboration of the conceptual architecture of the Information System;
- *User Interface Designer*, by supplying a structured business model, that includes the service design, *regulations* and *resources*, providing a user *task* context, to which are applied user-centered techniques targeting performance and testing;
- *Software Developer*, by providing a Software Architecture that allows understanding the dependencies and importance of each programmed class by relating each component with the business model artefacts.

The Goals Approach is presented using capital letters (GOALS), a situation also be applied to other method's names in order to facilitate recognition among Models and GOALS Modeling Concepts (both capitalized), *Techniques* (italic capitalized) and business and software development usual *concepts* (italic not capitalized) names. We present GOALS as a solution to model and trace business and software, and as a contribution for the communication among enterprise developers, targeting the improvement of their productivity, namely concerning the production of better software and the identification of new opportunities of cooperation and improvement in iterative (bigger or smaller) increments, as our background and motivation.

1.1. Origins

The combination of views and models is one of the originalities of the GOALS Approach. The approach relies on existing business and software languages to describe its underlying concepts. Our approach is mainly based on WISDOM [Nunes, 2001] which supplies the core software engineering process and the User Interface design focus [Nunes & Cunha, 2001], and on PROCESS USE CASES [Valente & Sampaio, 2007] (which is also a WISDOM extension), that establishes a structural relation between the business and software models using *essential use cases* [Constantine & Lockwood, 2001]. The GOALS Approach applies CTT (Concur-Task Trees) [Paternò, 1999] in two-steps: (1st step) decomposing *essential use cases* in terms of User Intentions using a Task Model; and (2nd step) decomposing User Intentions in User Interactions applying the BEHAVIOR DRIVEN DEVELOPMENT (BDD) [Chelimsky et al., 2010]) method, describing the system behavior, and designing the User Interface using modeling tools of the ACTIVITY MODELING (AM) method [Constantine, 2006]. The structuring of the system behavior and the internal structure of the Software Architecture comes from the WISDOM *Software Architectural Technique*, that provides support for the User Interface and complements the Domain Model [Booch et al., 2005], for the specification of the Information System. GOALS is structured on DEMO [Dietz, 2006] as a mean to specify how *business processes* human interaction is supported by GOALS User Tasks (*essential use cases*), including DEMO Action Rules and Object Classes, which are complemented with the Interaction Space (IX) concept, in order to support human interaction representation beyond the standard in-person interaction, and for more than two *actors*.

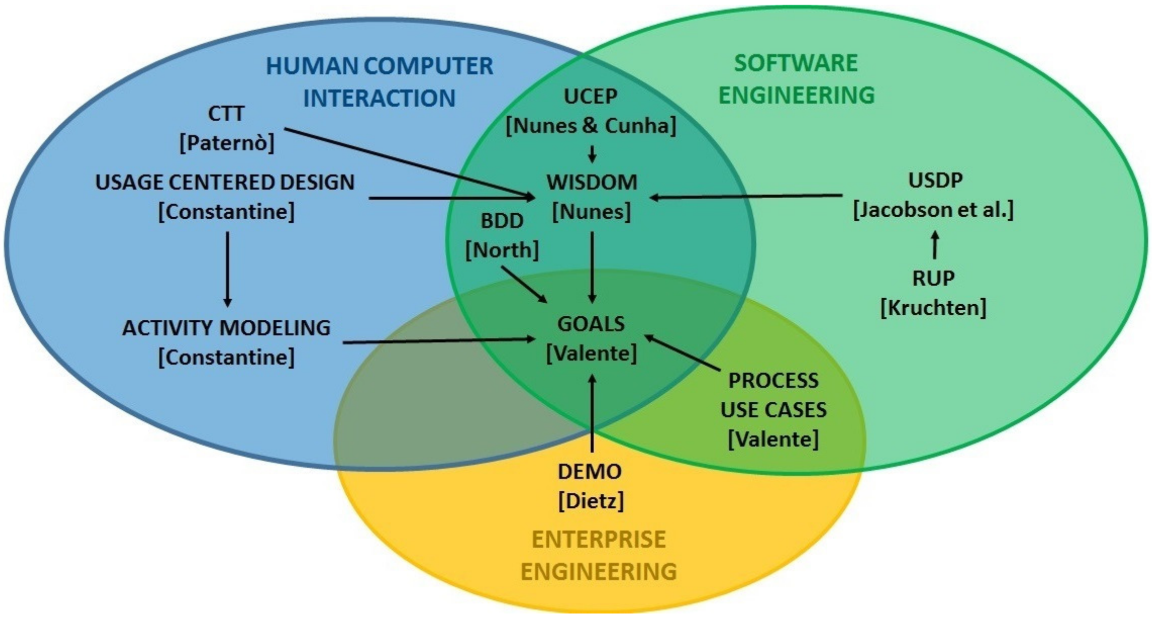


Figure 1. GOALS Origins.

Figure 1 summarizes the methods from which GOALS is originated, firmly based in SE by means of RUP [Kruchten, 2004] and USDP [Jacobson et al., 1999], and complemented with a (user-centered) Human-Computer Interaction (HCI) perspective since the beginning by means of the WISDOM method, which is complemented with ACTIVITY MODELING (AM) techniques concerning the design for performance, later complemented with BDD. This logic was extended to the Enterprise Engineering (EE) domain by the means of the PROCESS USE CASES method, and based on DEMO concerning Enterprise Architecture specification.

1.2. Motivation

The main motivation for the establishment of a single language is the improvement of the Software Development Process (SDP) performance. A unified business and software modeling language and method opens a space of communication for those who carry on the practice of modeling the business and the software. We formalize that a software development team includes the *Software Architect*, the *User Interface Designer* and the *Software Developer* stakeholders. They have the responsibility of deciding the process, method, techniques and implementation models of the Information System. The project team is only complete with the stakeholders which are responsible for the business, which are the *Business Owner*, the *Service Designer*, and the *Business Process Manager*, whom have the responsibility of providing the *resources*, specifying (by modeling) and implementing (based on software or not) the *business processes* of the enterprise. We assume that any of the team members may accumulate more than one role, and that a role may be composed by more than one member (a team) with distinct backgrounds. The objective is that all may use the same models and language so that they may understand the BPI and its impact in the customer and collaborator relation, and complementarily the impact of the improvement over the structure of the Information System.

By means of applying common development techniques, the business and software modelers may understand the implications of their decisions in the other modeler's work. And by using the same method, they can also understand the logic that leads from one model to the other, and from one concept to the other, even if they are working at the top-level concept (the *goal* of the *business process*) or the detailed User Interface component that allows its faster execution. By using the same language (and notation), project meetings may be faster, and by using the same models, the documentation may be reduced and will also be more easily recognizable, since it will be recurrently reused for the purpose of the ongoing "project dialogue". As such, the produced models also have value for more stakeholders. By using recurrently the (same) notation, the semiology of the development practice may also benefit from the stability provided by the usage of common business and software symbols, and therefore may more easily be assimilated by the patterned enterprise development practices which may already exist. Beyond the objective of organizing and facilitating the process of transferring business *requirements* and transforming them in a software architecture, also exists the objective of being able to understand how much working-time an Information System will take to be finished. Those metrics depend directly on the design of the business and the software parts which will be implemented. Yet, its measurement and therefore its estimation, can only be attained if the SDP is properly stabilized concerning the production of software as a business-support for the enterprise, probably optimized by means of the usage of a single enterprise modeling language.

These are the main inspirations that drive our research, which also reflect a desired state of evolution of the Software Engineering (SE) discipline, from an ancient state of chaos [Morgenshtern et al., 2007], to a more stable but still unproductive phase, that reports that full software development project success is only around 20% (within budget and time) [The Standish Group, 2014], despite the advances introduced by agile methods [VersionOne.com, 2017], and by the user involvement in the development process [The Standish Group, 2013]; to a phase of maturity where there is the increased awareness on what may be the Return of the Investment (ROI) based on the business and software design decisions as an ultimate challenge [Valente et al., 2015]. Our motivation is now expressed in terms of the business and software modeling contributions which we introduce as follows.

1.2.1. Business Modeling Contribution

One of the key factors for project success is reaching team consensus over the BPI concerns, which is easier if the team understands the relation of each of the business with each of the software modeling concepts, in order to promote a faster and more effective understanding. By using concepts which are originated from the scientific domains usually involved in in-house software development, with adaptations in terms of: scope, structure and relation (between concepts); our language introduces fruitful changes to the software development practice. The first change, of structure, is that *business processes* should be named after their *goal*, as it is the enterprise-recognizable *business concept* that reflects the undergoing objective, helping focusing the target of software implementation. The second change, of scope, concerns modeling the activities of the *business processes* using *essential use cases* [Constantine & Lockwood, 2001], which are defined as "*a complete task performed by one actor*", in order to withdraw margins of uncertainty concerning *requirements* elicitation, especially in terms of number, establishing a simplified and comprehensive (an "*actor task*-sized") relation between business and software models. The third change, of relation of concepts, is the inclusion of the Interaction Space (IX) as the space where communication between *actors* happens, and where specific *regulations* are applied over the management of certain *data*. The relation between *goals*, *essential use cases*, IXs, *regulations* and *data*, is the "key" that enables the composition of a business model that allows rationalization about the enterprise development problem on a single diagram, and serves as the "door" to the specification of the Information System which will support the BPI, as those concepts will be detailed, complemented and included as a part of the final Software Architecture. By means of using the same language in a single top-down analysis logic (from *business process* to *data* set), we are also creating conditions to focus on what are the more important concepts that must be transferred from the business team to the software development team. As a result, less work may be lost in the transfer, and there is no need for translation efforts, contributing to increase the performance of the EDP.

The main contribution for the business modeling is related to the specification of a Business Architecture that can receive software architectural support independently of the type of design of its *business processes*. The Business Architecture (represented by the Enterprise Structure model), becomes part of the Software Architecture, composing the Enterprise Architecture. The Enterprise Structure includes the Interaction Space (IX) as the "aggregator" of *data* and *regulations* which are at the disposal of the *tasks* which may be performed in those logical spaces. The IX allows the representation of the interaction of one, two or more *actors*, and the representation of the physical space where they interact, and which actions are performed. The IX is then included in the Software Architecture as the component that ensures the application of the business *regulations* (Business Rules, BR) and the usage of the *data* (Data Entities, DE). The IXs, BRs and DEs are therefore shared concepts of the Business and Software Architectures, which also represent a contribution to the SE domain by means of the inclusion of the BR concepts as the business-specific concept of the Business Logic layer of the MVC pattern of the Information System.

Although the proposed modeling structure may be complemented in order to measure *business process* performance, using e.g. Key Performance Indicators (KPIs), these aspects are out of the scope of current version of our approach, as its main target is the provision of a tool that introduces enterprise enhancements which advantage is not in question of doubt, or in other words, are "structural" to the enterprise.

1.2.2. Software Modeling Contribution

The SDP performance can be strongly increased if the business model of the enterprise is structured on components which are meaningful to software practitioners, and which may be further detailed in order to produce the Information System internal structure. This is possible by means of the detail of the human activity “down” to the level of interaction, and may be achieved by using Human-Computer Interaction (HCI) techniques, representing an added value for both business and software modelers, as they will be able to identify the Information System components that support the *business process*-driven human mental and physical (partial, working) activity. Once this is achieved, the dialogue between business and software modelers, including *User Interface Designers*, may be fluent and potentially more fruitful based on a better understanding about the business-problem and software-solution under analysis, prospecting a project team faster agreement concerning the operationalization of the BPI. By integrating the Enterprise Structure (Business Architecture) as the conceptual “back-bone” structure of the Software Architecture, the received components (instantiations of concepts), only have to be complemented with software-specific components to be conceived as the final Information System implementation-independent architecture (the Software Architecture). As the produced structure is directly related to an MVC implementation model, a standardized straight-lined process may be assembled, based on a methodical implementation process easier to carry out, eventually with a closer support from business modelers.

As our focus is the organization and coordination of business and software modeling, the measurement of the development performance is out of the scope of the current version of our approach, as the organization and automation of the SDP can be seen as a pre-requisite of software success, aiming in the first place the effectiveness of the process, and in second place its efficiency.

1.3. Research Question

The research question of our thesis concerns the establishment of a language and method to produce the artefacts which are needed for in-house software development. The question is considered as holistic in the perspective that, with a single language, one must be able to develop software solutions to all the business activities that manage enterprise *data*. The pertinence of the question exists because actual software development methods do not cover all the business and software modeling relevant aspects maintaining traceability, and because the ones that do are not supported by a development method. The research question is formulated as follows.

Q1: *Is it possible to establish a modeling language that can be applied by a single method which is representative of the organized business activity and its supportive software system*?

The research methodology was oriented by means of the usage, generalization, conciliation and exclusion of scientific research methods and concepts from distinct research domains concerning software development, during software practice in the University of Madeira (UMa), a medium-sized enterprise, between 2002 and 2011. From this practice, an initial version of GOALS was published as GOALS SOFTWARE CONSTRUCTION PROCESS [Valente, 2009]), which (summarizing) was an extension of the WISDOM method [Nunes, 2001] by means of the PROCESS USE CASES method [Valente & Sampaio, 2007], using the UNIFIED MODELING LANGUAGE (UML) [Booch et al., 2005].

The research method is originated from a logic that results from software development practice with academic support as needed. This first application resulted in the elaboration of the Software Architecture for a "major" BPI in the University of Madeira that included 5 "core" *business processes* (from the pedagogical service: "distribution", "scheduling", "preparation", "lecturing" and "evaluation"). The Software Architecture was produced using WISDOM, and related to the enterprise *business processes* by means of PROCESS USE CASES, and its User Interface was designed using CANONICAL ABSTRACT PROTOTYPES [Constantine et al., 2003], which received support from the HYDRA framework [Costa et al., 2007], for the implementation of the User Interface using the remaining identified (Business Logic and Database) Software Architecture components. The application was replicated for the implementation of 7 *business processes* (course "course inquiry", "business unit management", "facilities management", "facilities access control", "academic billing system", "student credits accreditation", and "student academic background"), and leaded to the establishment of GOALS as an internal SDP standard. The method was further used for 3 more BPIs ("course accreditation", "student's academic requests" and "voting system"), before it was formalized starting in 2012, and continued to be applied as a form of validation of its models in other BPIs. Since traceability was found to be an unsolved problem in both Enterprise Engineering (EE) and SE domains, the method was formalized concerning the methodical application of its development techniques in consecutive steps.

As our research question targets business and software Information Systems modeling traceability, our hypothesis reflects the establishment of a language of shared concepts as a solution, as follows.

H1: *It is possible by means of a cross-consistent relation of business and software concepts, and by modeling the business human activity and relating it to software-specific components of the software model by means of the application of architectural patterns.*

The strategy that sustains H1 is mainly based on the identification of the spaces where users carry on their *tasks*, aggregating in a logical relation: the *tasks*, the business *regulations* and the *data* that apply in each case. These spaces are called as Interaction Spaces (IX), and where found, in our opinion, to be the missing part of the business representation of existing methods, and it is based on it that we apply the remaining techniques that bridge business and software modeling. With the purpose of supporting H1, and beyond the validation provided by its recurrently successful application, GOALS also received distinct forms of validation, concerning: (1st) its performance, (2nd) its business structure, and (3rd) its conceptual structure:

- First, a study of the application of the method was carried out for five BPIs [Alves et al., 2013], using the iUCP method [Nunes et al., 2011] for post-project estimation and comparison of software development effort, revealing a performance stability that allowed the identification of: *requirements volatility*, *software framework* (change), and *project type* ; as aspects that add more (negative or positive) impact to the development productivity than considered by the initial UCP method [Karner, 1993]. It led to the proposal of a new formula which would reflect those aspects with higher accuracy. The statistical evidence was observed for a seven BPI sample [Alves et al., 2013, p. 4] (P1 to P5, "facilities access control"; "academic billing system"; "students credits accreditation"; "course accreditation"; and "student's academic requests"; and two BPIs (P6 and P7) from other stable software development environments, using other methods.

- Secondly, it was also evaluated concerning its business conceptual structure, by means of the comparison with the DEMO [Dietz, 2006] *business process* equivalent representation [Valente et al., 2016b], highlighting the inclusion of the IX as the solution to provide software architectural support for the (DEMO) *business process* Transactions, modeled with *essential use cases* to represent human activity, and including the IX as the element that provides support for that activity by ensuring *regulations* and *data* conformance, for two of more *actors* performing their *tasks* (*essential use cases*) in remote or in-person interaction.
- Thirdly, the GOALS language structure was validated concerning the cross-consistency of concepts [Valente et al., 2017] (in a 10 step method, now presented in 8 steps), and architectural restrictions concerning the organization of the business and software models. The validation (extended) is presented in Section 6.1. Cross-Consistency. The method was also published concerning its agile Human-Centered Software Engineering (HCSE) perspective [Valente et al., 2016a], as a trade-off between agility and traceability (architectural control) of business and software representations.

Based on the previously referenced work, the formalization of our method when through distinct phases, which are described as follows:

- **Effort Estimation and SDP Productivity Evaluation**. A study was carried out in order to evaluate the *use case* as a "measure of effort" concerning software productivity using the GOALS method. With a sample of five (University of Madeira) projects, two projects from other enterprises were added, providing a (statistically workable) heterogeneous sample of seven. From the analysis of the factors that mostly affected each project (*requirements volatility*, *software framework* (change), and *project type*), it was possible validate a new formula effort of the iUCP method, by giving those factors a (bigger) weight that would reflect their real impact. The study received the honor of "Best Full Paper Award" from the conference.
    - (Full Paper) [Alves et al., 2013] Alves, R., Valente, P., Nunes, N. *Improving Software Effort Estimation with Human Centric Models - A comparison of UCP and iUCP accuracy*. In: Proceedings of 5th ACM SIGCHI Symposium on EICS '13, London, England. pp. 287-296. (2013)
- **User Experience and Service Design**. A study was carried out, directed to the HCI and SE industry practitioners (sample of 97 questionnaires), in order to understand how the User Experience (UX) concerns were considered within the software development practice. It allowed understanding how strongly BPI ("*business reorganization*") projects are related to the (paper) prototyping of the User Interface, namely in the phases of requirements elicitation, design and testing. And confirm that refactoring is strongly correlated to requirements evaluations (introduction or change).
    - (Full Paper) [Alves et al., 2014] Alves, R., Valente, P., Nunes, N. *The State of User Experience Evaluation Practice*. In: Proceedings of the 8th Nordic Conference on Human-Computer Interaction, pp. 93-102. (2014)
- **Intention to Improve Software Estimation**. A position paper was published, declaring the importance of introducing changes to the SDP that would allow measuring the ROI of a project after requirements elicitation and before software implementation. And that a business driven system design should dictate the effort as a target for the accuracy of estimation effort, aiming the production of better software.

  - (Short Paper) [Valente et al., 2015] Valente, P., Aveiro, D., Nunes, N. *Improving Software Design decisions towards enhanced Return of Investment*. In: Proceedings of ICEIS 2015, Barcelona, Spain. pp. 388-394. (2015)
- **Intention to Bridge EE and SE**. The intention of bridging EE and SE was discussed as a project in a Doctoral Consortium, including the actual business model, the importance of defining an enterprise-sized development process (EDP) that includes the software development, and how effort estimation may be based on business and software models in the future (by weighting architectural components).
  - (Doutoral Consortium) Valente, P., Aveiro, D., Nunes, N. (2015). *Bridging EE and SE: The Goals Approach*. Enterprise Engineering Working Conference. https://www.researchgate.net/publication/283289437_Bridging_EE_and_SE_The_Goals_Approach
- **Business Model based on DEMO**. The GOALS business model and its relation with the DEMO model was published, and how the "bridge" to Software Architecture could be achieved by means of a User-Centered perspective, applying the method with a total of 10 steps (now reduced to 8).
  - (Short Paper) [Valente et al., 2016b] Valente, P., Silva, T, Winckler, M., Nunes, N. *Bridging Enterprise and Software Engineering through a User-Centered Design perspective*. In: Proceedings of the Web Information Systems Engineering (WISE) 2016 Conference. (2016)
- **GOALS and Systems Development**. The GOALS method was published, targeting its meta-model and concepts cross-consistency validation. Targeting model-driven in-house software development, focus is also made on the technology that may be applied to implement each Software Architecture component.
  - (Full Paper) Valente, P., Silva, T, Winckler, M., Nunes, N. *The Goals Approach: Agile Enterprise Driven Software Development*. In: Proceedings of the Information System Development (ISD) 2016 Conference. (2016)
  - (Extended Full Paper) [Valente et al., 2017] In: Lecture Notes in Information Systems and Organisation, 22. (2017)
- **GOALS and Human-Centered Software Engineering**. The GOALS method was published focusing the proposal of a trade-off between business architectural control and agile (user) *task*-sized (User Task (UT)) improvement efforts, and the need to induce the multi-disciplinary work within enterprises targeting project success.
  - (Full Paper) [Valente et al., 2016a] Valente, P., Silva, T, Winckler, M., Nunes, N. *The Goals Approach: Enterprise Model-driven Agile Human-Centered Software Engineering*. In: Proceedings of the HCSE+HESSD 2016 Conference. (2016)

Our research was complemented with the comparison of our method with existing methods that address the business and software models relation. The original scientific domains (EE, HCI and SE) where complemented with Business Process Management (BPM), Service Design (SD), Requirements Engineering (RE), and Service Oriented Architecture (SOA) methods, for the purpose of comparison in terms of the coverage of the enterprise Information System development problem using the Zachman Framework (ZF) [Zachman, 2008] which is presented in Chapter 2. State of The Art, and in terms of the coverage of the business and software modeling concerns, presented in Chapter 6. Validation.

1.4. Thesis Organization

The thesis is organized in seven chapters, which we introduce as follows.

- **Chapter 2. State of The Art**, presents the state of the art concerning business and software modeling traceability, focusing on the business model conception, and how *requirements* are elicited targeting the software model architecting;
- **Chapter 3. The Goals Language**, presents the structure of our language, its concepts, stereotypes and definitions, including their meta-model, an example, and their origin;
- **Chapter 4. The Goals Process, Method and Techniques**, presents the process, method and techniques, including two case studies based on a real software development projects: ("iKiosk") focusing the method application, and; ("Star Project"), focusing the decisions taken during the application of the method.
- **Chapter 5. Goals EA Template**, presents how our method may be applied using a software tool available in the market (Enterprise Architect);
- **Chapter 6. Validation**, presents the validation of the language by means of its internal cross-consistency, and its power of expression by means of comparison with other methods;
- **Chapter 7. Conclusions**, presents our conclusions and future work.

2. State of The Art

The scope of our language and method is the modeling of the business and the supporting Information System, targeting the specification of a business-traceable Software Architecture suitable for in-house software development. In this scope, we find the intention of "bridging" business and software models in methods originated from Enterprise Engineering (EE), based on business modeling, in Software Engineering (SE), targeting software modeling, and in Human-Computer Interaction (HCI), targeting software system behavior modeling for the support of human activity. These domains specify the *requirements* of an enterprise Information System that contains *data*, implements business *regulations*, and provides adequate behavior i.e. an Information System suitable for the daily enterprise operation. EE, SE and HCI, are in the origin of our proposal, and are complemented with the Service Design (SD), Business Process Management (BPM), Requirements Engineering (RE) and Service Oriented Architecture (SOA) domains, to extend these *requirements* to consider enhanced service and business process experience and connectivity. Each domain is introduced as follows.

In the business perspective, the SD domain represents a very wide scope in terms of Information Systems *requirements*, which refers not only to the usability of the software, but also to the (emotional) User Experience (UX) in interaction with the business [Alves et al., 2014]. EE methods model the enterprise by means of elaborating Enterprise Architectures that include *actors* and *business processes*, and also *transactions*, *regulations* and *data* structures. The structures may be complemented with Organizational Operative Systems (OOS) to generate operational software, and with software development methods concerning Information System engineering. Closer to the SE domain, in the field of automatic code generation, BPM methods produce Workflow Management Systems (WfMS) from *business process* models, usually integrating business *regulations* in the final Software Architecture, which has a parallelism with the EE approaches that produce working software.

In a human-centric perspective, HCI methods are not holistic business representations, however, they consider the context of interaction and also what the user expects from the system, usually with structured feedback, promoting the user performance and UX, besides and beyond ensuring usability. Those methods also produce valid in-house software architectures in the cases when they also include business *regulations*, and produce Database valid information by means of *business concepts* that model the domain of the Information System.

In an Information System engineering perspective, RE strongly addresses the problem of traceability, observing it as the consequence of eliciting *requirements* from business models, independently of the used form (e.g. *goal*, *use case*, aspect or feature); techniques which are crucial in terms of elicitation capability and accuracy, as *requirements* generate the complexity managed in a SDP. In SE, existing methods elicit business *requirements* and apply transformations that produce Software Architectures. These are Model-Driven Architecture (MDA) methods, which are mostly derived from the RATIONAL UNIFIED PROCESS (RUP) [Kruchten, 2004], based on the responsibility which is isolated in *use cases* as a source of *requirements*. SOA methods, model the service and produce web-service structures that implement business-to-business. The methods do not model the User Interface, however, they can be used for enterprise Information Systems if its conception is provided by an alternative method. SOA establishes a relation between the business' needed functionality and the system Business Logic, ensuring traceability.

The more representative methods of each domain will be analyzed concerning Information System conception and the requirements defined by the Zachman Framework [Zachman, 1986], with the purpose of understanding how comprehensive each method is, and how business and software traceability is achieved. We present the Zachman Framework as follows.

2.1. The Zachman Framework for Enterprise Architecture

The Zachman Framework (ZF) [Zachman, 1986], was conceived with the purpose of understanding the scope of what must be controlled within an enterprise, so that all of its parts are documented to provide a holistic view of the enterprise architecture which may be used as a base of its management. The ZF scope of analysis of the enterprise includes its business and Information System, divided in levels, from the more abstract to the more concrete perspective: "Scope", "Concept", "Logic", "Physics", "Technology", and "Product", relative to the roles of "Planner", "Owner", "Designer", "Builder", "Implementer", and "Operator". These are crossed with the interrogatives: "What?", "How?", "Where?", "Who?", "When?", and "Why?"; specifying the enterprise architecture independently of the used method and notation. The ZF provides an ontology about the object under study, which in our case is the modeling of the business and software systems of the enterprise, the initial purpose of the ZF, despite the fact that it may be used for every type of object (besides the enterprise: a project, an airplane, a building, etc.) [Zachman, 2008].

Table 1. The Zachman Framework (ZF) Matrix.

	What	How	Where	Who	When	Why	
Planner	**Inventory Identification**	**Process Identification**	**Distribution Identification**	**Responsibility Identification**	**Timing Identification**	**Motivation Identification**	**Scope**
	List: Inventory	List: Process Types	List: Distribution	List: Responsibility	List: Timing Types	List: Motivation	
Owner	**Inventory Definition**	**Process Definition**	**Distribution Definition**	**Responsibility Definition**	**Timing Definition**	**Motivation Definition**	**Concept**
	Busines Entity	Busines Transform	Busines Location	Busines Role	Busines Interval	Busines End	
	Busines	Busines	Busines	Busines Work	Busines Moment	Busines Means	
Designer	**Inventory Representation**	**Process Representation**	**Distribution Representation**	**Responsibility Representation**	**Timing Representation**	**Motivation Representation**	**Logic**
	System Entity	System Transform	System Location	System Role	System Interval	System End	
	System	System	System	System Work	System Moment	System Means	
Builder	**Inventory Specification**	**Process Specification**	**Distribution Specification**	**Responsibility Specification**	**Timing Specification**	**Motivation Specification**	**Physics**
	Technology Entity	Technology	Technology	Technology Role	Technology	Technology End	
	Technology	Technology	Technology	Technology Work	Technology	Technology Means	
Implementer	**Inventory Configuration**	**Process Configuration**	**Distribution Configuration**	**Responsibility Configuration**	**Timing Configuration**	**Motivation Configuration**	**Technology**
	Tool Entity	Tool Transform	Tool Location	Tool Role	Tool Interval	Tool End	
	Tool Relationship	Tool Input/Output	Tool Connection	Tool Work Product	Tool Moment	Tool Means	
Operator	**Inventory Instantiation**	**Process Instantiation**	**Distribution Instantiation**	**Responsibility Instantiation**	**Timing Instantiation**	**Motivation Instantiation**	**Operation**
	Operations Entity	Operations	Operations	Operations Role	Operations	Operations End	
	Operations	Operations	Operations	Operations Work	Operations	Operations Means	
	Material	**Process**	**Geometry**	**Instructions**	**Timing**	**Objetives**	

The crossing of the ZF conceptual levels with the interrogatives is presented in Table 1, a matrix of 36 cells in which are specified the modeling requirements that should be applied. Each cell suggests models based on the crossed concepts. The "Scope" level (at the top of the ZF) suggests the elaboration of lists of "Identification" for: "Inventory", "Process", "Distribution", "Responsibility", "Timing" and "Motivation". The underneath levels of: "Concept", "Logic", "Physics", "Technology", and "Product", maintain the same logic, but now crossed with the modeling concepts of: "Definition", "Representation", "Specification", "Configuration" and "Instantiation" (replacing "Identification"), getting gradually closer to implementation.

This logic allows the representation of the enterprise architecture, to which we introduce a change concerning the specification of the "Where" column and the concept of "Distribution", by means of extending it to the logical location, adding it to the original proposal (which initial name is "Network"), which refers to physical location, allowing it to also support virtual enterprises. Thus, this column has enhanced meaning if both physical places and logical spaces of the enterprise system are also represented in the architecture, whereas, the *data* network, should only be considered at the "Operation" level, which represents the materialization of the enterprise, and which is out of the scope of our analysis. The more representative methods of each scientific domain will be matched with the ZF, according to the literature review, which we present as follows.

2.2. Business-driven Modeling Methods

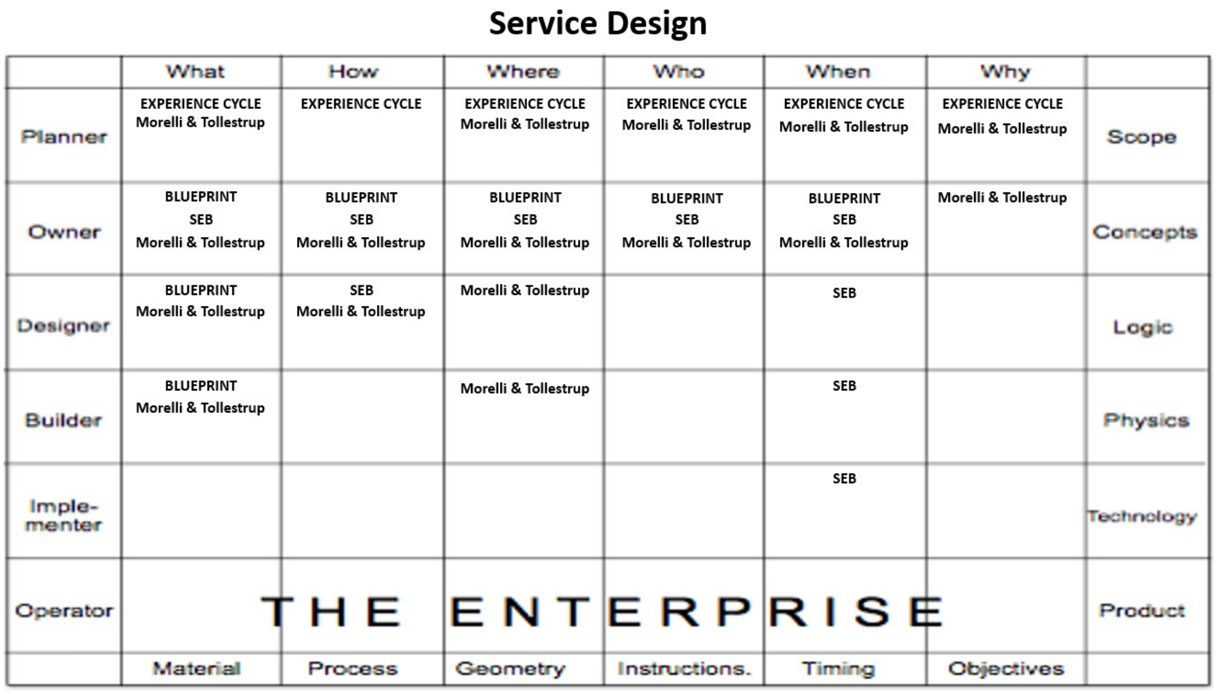
Business models aim at structuring the business with humans and system as equivalent and complementary parts towards the operationalization of the enterprise strategy. In the business modeling perspective, the organization of the parts usually relies on advanced technology from which the Information System conception must be tailored in order to fit the enterprise performance needs. The Service Design (SD), Enterprise Engineering (EE), and Business Process Management (BPM) more representative methods are presented as follows.

2.2.1. Service Design

Service Design (SD) started as a discipline of marketing and management of the service, where the specification of products and services assumed an important role in the modeling of the service and customer interaction, demanding levels of usability from Information Systems that may support its customer-oriented strategy. The EXPERIENCE CYCLE [Dubberly & Evenson, 2008] is a customer oriented method that defines a specific strategy for the service-customer relation, in which the objective is to "Orient" (the customer), "Interact" (with the customer), so that the customer will probably wish to "Extend & Retain" (the experience), "Advocate" (about it), and "Connect & Attract" (other customers). The BLUEPRINT [Shostak, 1982] was the pioneer method, relating services and products for the purpose of designing the service as a whole. The BLUEPRINT establishes a relation that can be used for different purposes, including the development of Service Oriented Architectures (SOA) [Becker et al., 2008] (SOA methods are analyzed in Section 2.4.3. Service Oriented Architecture). In the BLUEPRINT, the logic that relates processes and *resources* is the rationale behind the way of thinking, analyzing and delineating the business. This logic by itself is however far from the level of detail needed to design the proper User Interface that satisfies customers over time. SD methods also express human intention and activity with detail.

The SERVICE EXPERIENCE BLUEPRINT (SEB) [Patrício et al., 2008], is a method based on the BLUEPRINT that decomposes *softgoals* and *goals*, and further specifies the relation between the actions that the user takes in the dialogue with the enterprise system as his experience sequence (the route), in a similar way that *essential use cases* are decomposed in terms of user intentions and system responsibilities [Constantine & Lockwood, 2001]. SEB prevents the over-decomposition of *requirements* (too much granularity) before *task* analysis that may lead to solutions more complex than necessary. It also has the benefit of increasing the probability of producing a more complete User Interface, since when the service design is evolved there is the tendency to reduce the number of interactions (touch-points), maintaining or increasing the number of provided services.

The SD challenges can also be considered as a reference concerning usability as they also design the user's (customer) emotions. The MORELLI & TOLLESTRUP's [Morelli & Tollestrup, 2006] method, elaborates a motivation matrix compliant with the user experience levels demanded by the market [Pine & Gilmore, 1998], that may eventually be used to design User Interfaces with a high-level of performance. This method is an example of a valid high-level requirements modeling approach targeting the specification of Information Systems, however, the modeling of the internal structure of the system, or even the User Interface, is out of the scope of the method beyond the proposed identification of *use cases*.



Service Design

	What	How	Where	Who	When	Why	
Planner	EXPERIENCE CYCLE Morelli & Tollestrup	EXPERIENCE CYCLE	EXPERIENCE CYCLE Morelli & Tollestrup	EXPERIENCE CYCLE Morelli & Tollestrup	EXPERIENCE CYCLE Morelli & Tollestrup	EXPERIENCE CYCLE Morelli & Tollestrup	Scope
Owner	BLUEPRINT SEB Morelli & Tollestrup	BLUEPRINT SEB Morelli & Tollestrup	BLUEPRINT SEB Morelli & Tollestrup	BLUEPRINT SEB Morelli & Tollestrup	BLUEPRINT SEB Morelli & Tollestrup	Morelli & Tollestrup	Concepts
Designer	BLUEPRINT Morelli & Tollestrup	SEB Morelli & Tollestrup	Morelli & Tollestrup		SEB		Logic
Builder	BLUEPRINT Morelli & Tollestrup		Morelli & Tollestrup		SEB		Physics
Imple-menter					SEB		Technology
Operator		T H E	E N T	E R P	R I S E		Product
	Material	Process	Geometry	Instructions.	Timing	Objectives	

Figure 2. ZF Matching of the SD methods.

Figure 2 presents the matching of the SD methods concerning service elaboration, traceability and software development. It is possible to identify that the EXPERIENCE CYCLE and the BLUEPRINT methods cover the top of the conceptual logic of the ZF, providing insight of the logic of the business, the customer intention and the expected route of interaction. The Becker method focuses on functionality in a SD branch that directly relates to SOA, and the MORELLI & TOLLESTRUP's method focus on motivation not however providing a software architecture. The SEB method relies on *essential use cases* decomposition based on a line of interaction avoiding *use cases* indefiniteness in the representation of the customer relation.

2.2.2. Enterprise Engineering

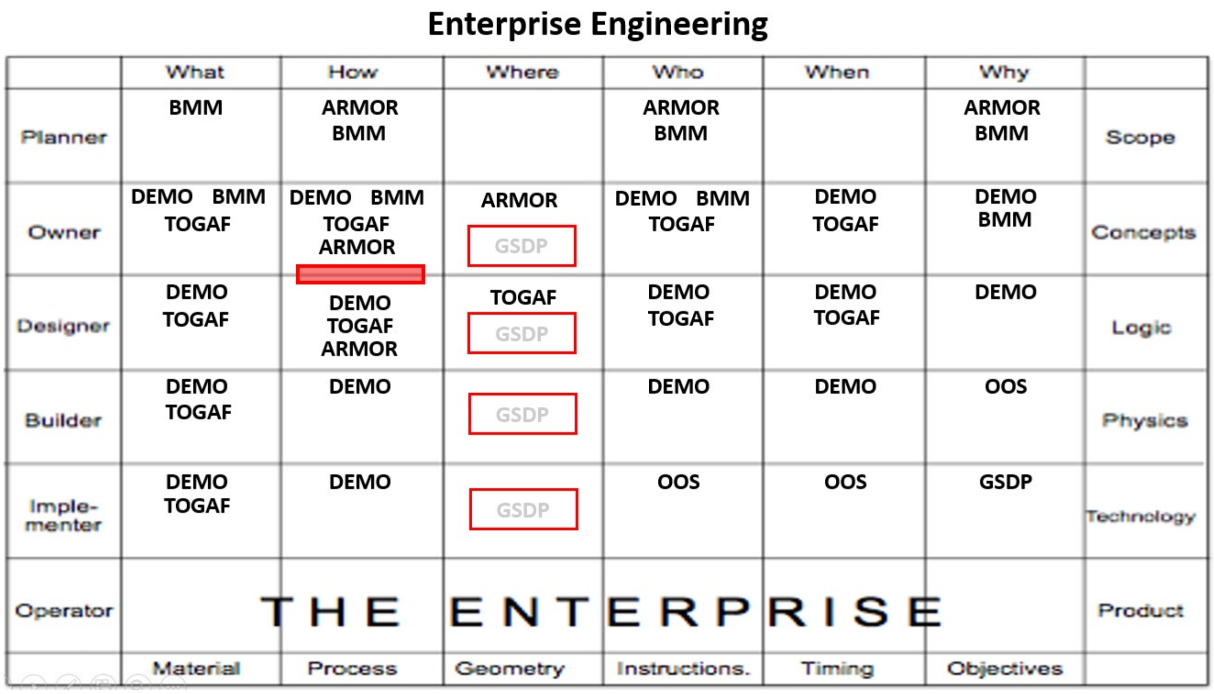
Enterprise Engineering (EE) has emerged as a discipline for the application of knowledge, principles, and techniques to the analysis, design, implementation and operation of all elements associated with an enterprise. Existing methods derive software solutions from business models in the process of architecting and engineering the parts of an enterprise that suit its implementation and governance. The development of an Enterprise Architecture is a usual practice of EE methods, which widely vary in scope and form of representation of the enterprise business, being more market oriented and less software-architectural, or more Information System development oriented and less human-centric.

The Business Motivation Model (BMM) is an enterprise strategy model. BMM targets the specification of the reasons "why" the enterprise acts towards a given "objective", and how it should achieve it. The model serves as a support in order to understand the existence of the *business processes* of the enterprise, and what are the desired results, according to the enterprise strategic vision. BMM reflects only a part of what is the business model, since *business processes* are not designed, as the concerns are more related to the courses of action as a whole, and the related business rules and policies conception, including their relation to internal and external motivators. BMM does not consider the software model.

The Open Group Architecture Framework (TOGAF) [The Open Group, 2011], is a method that predicts the elaboration of a business, a *data*, a technology, and an application architecture. TOGAF targets the specification of the enterprise, and the identification of its building blocks using the ARCHIMATE language [The Open Group, 2012]: "Application Component", "Application Function", "Application Interface", "Application Service", and "Data Object". These architectural components may be related to equivalent business components, including "Business Services" and "Business Processes", and also to infrastructure (e.g. software server) components. However, the identified building blocks do not provide enough detail in order to architect the software system, and as such, TOGAF does not establish a relation that ensures traceability at the level of the implemented component. Yet, as the application parts are identified along with the *business processes*, other SE methods may be used in order to model the software, such as the ARMSTRONG method [Armstrong et al., 2013], published by The Open Group. The resulting transformation of the method uses the "Business Interaction" to derive "Business Use Cases" between the involved *actors*, and then use an Activity Diagram to represent the *business process* of that interaction, with which it derives the functionality that must be implemented by *use cases*. In our perspective, this method also leaves gaps in the relation between the business and software modeling relation, namely: what is the relation between the "Business Use Cases" and the *business process* "activities", and what is the relation between the "activities" and the *use cases*, also skipping the design of the User Interface. Moreover, TOGAF provides a set of modeling tools and documentation which we do not find as suitable for in-house software development, and observe it as a potential threat to the efficiency of the SDP due to management time overhead.

DEMO [Dietz, 2006] is an enterprise engineering method that produces valuable business and software artefacts concerning Information System conception. Software development is usually prospected using the GENERIC SOFTWARE DEVELOPMENT PROCESS (GSDP) [Kervel et al., 2013], which derives software specifications from the DEMO enterprise ontology to a system ontology. However, DEMO and the derived methods do not contemplate the User Interface design, since the user *task* model does not provide sufficient detail besides the ("coordination") acts ("request", "promise", "state" and "accept"), of a *transaction* between two *actors*. Another approach to Information System conception is the software operationalization by means of ORGANIZATIONAL OPERATING SYSTEMS (OOS) [Páscoa & Tribolet, 2015], which is also based on DEMO models, in order to deploy operational software. Yet, as DEMO does not produce enough information for User Interface specification besides the *business process* work flow and the coordination acts, the usability of the system cannot be ensured, resulting in conceptions that need User Interface adjustments in order to be usable [Hintzen et al., 2014].

ARMOR [Engelsman & Wieringa, 2012] is a method that adapted concepts from other methods, namely TOGAF [The Open Group, 2011], KAOS [Lamsweerde, 2000], i* [Yu, 1995], TROPOS [Morandini et al., 2014] and BMM [Object Management Group, 2007] in order to establish the specification of an Enterprise Architecture. The ARMOR architecture specifies the business in terms of stakeholder's *goals*, including the "influence" to which they are subject to, the possible "conflicts", and their decomposition. From the decomposition of *goals*, ARMOR derives "requirements" and then establishes a relation of one-to-one with the "Architecture Component" concept. There are more EE methods relative to enterprise architecting, like the LIVING MODELS [Breu et al., 2011] and the ZIKRA [Zikra, 2014] methods, which propose the formal design of *business processes* and software components, however, not with enough detail and without a well-defined software development method.



Enterprise Engineering

	What	How	Where	Who	When	Why	
Planner	BMM	ARMOR BMM		ARMOR BMM		ARMOR BMM	Scope
Owner	DEMO BMM TOGAF	DEMO BMM TOGAF ARMOR	ARMOR GSDP	DEMO BMM TOGAF	DEMO TOGAF	DEMO BMM	Concepts
Designer	DEMO TOGAF	DEMO TOGAF ARMOR	TOGAF GSDP	DEMO TOGAF	DEMO TOGAF	DEMO	Logic
Builder	DEMO TOGAF	DEMO	GSDP	DEMO	DEMO	OOS	Physics
Imple-menter	DEMO TOGAF	DEMO	GSDP	OOS	OOS	GSDP	Technology
Operator	THE ENTERPRISE						Product
	Material	Process	Geometry	Instructions.	Timing	Objectives	

Figure 3. ZF Matching of the EE methods.

Figure 3 presents the framing of the EE methods in the ZF. These complement each other to cover the layers "Conceptual" to "Technology". DEMO and TOGAF cover most of the modeling needs concerning the "What" and "How" columns, DEMO covers three layers of the "Who" and "When" columns, and is complemented in the "Technology" layer by the OOS, concerning software operationalization of *actors* and their coordination in *transactions*. The "Why" column is covered by the BMM (which is its main target), DEMO at a conceptual level, and the *regulations* implementation is left to the OOS and GSDP methods, in the "Physics" and "Technology" layers, respectively. As we do not find evidence of User Interface design in the presented methods, that chasm should be left to the GSDP method (in the "Where" column), but since no detailed user information is produced to from *transactions* and cannot find further evidence of its effectiveness in the literature, we also highlight that indefiniteness (with a red lined opaque rectangle). Still in the "Where" column, the TOGAF fills the "Designer" layer by means of the ARCHIMATE "Business Interface" and "Location" concepts. The ARMOR method distinguishes itself by means of modeling concern which may be matched win the "Scope" layer, namely by accessing stakeholder's concerns as its main target.

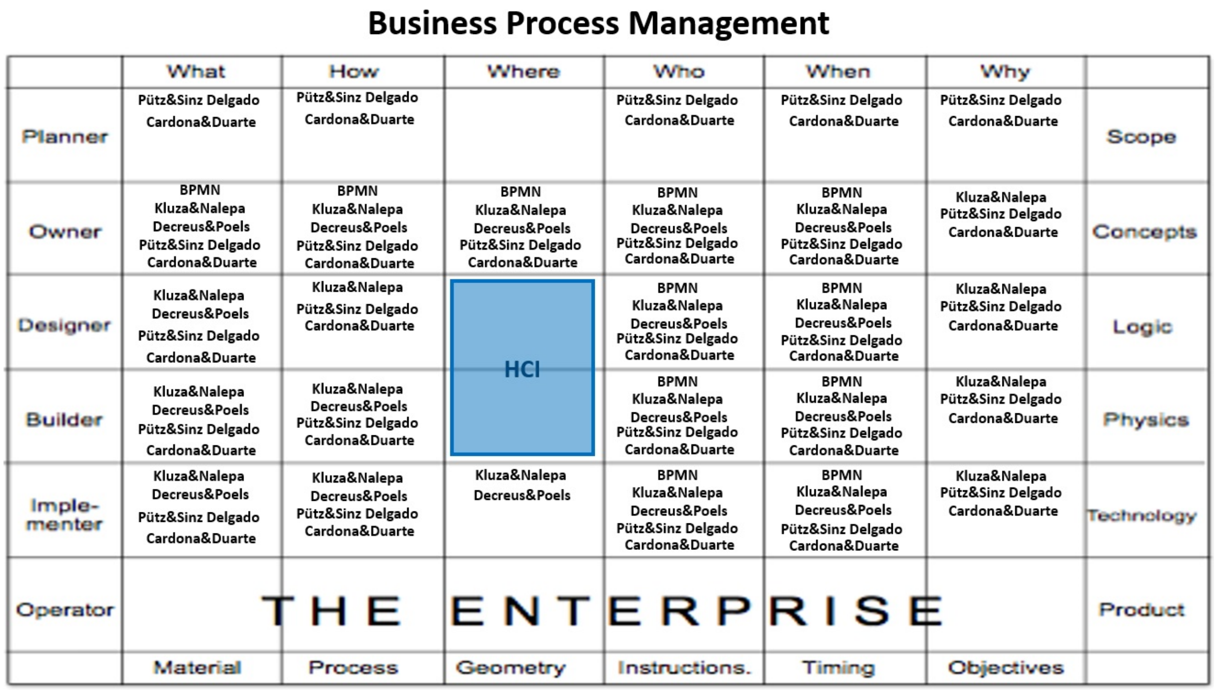
2.2.3. Business Process Management

Business Process Management (BPM), in the considered analysis scope, refers to the automation of *business processes*. BPM has evolved over time by means of the introduction of *business process* models and the execution of modeling languages [Ouyang et al., 2009], in order to produced Workflow Management Systems (WfMS). Software is designed according to the logic of *business processes*, producing User Interfaces that work, in which however usability is not addresses, as the information to achieve it is not present in the development process, reason why it is impossible to expect adequate behavior from the system without further complementing the methods with a user-centered perspective. The Business Process Model and Notation (BPMN) [Hagen, 2010], is the reference language on which rely most of the execution languages that implement workflow standards [Garcês et al., 2009]. BPMN is a *business process* modeling language, which however does not have a method to apply it in order to produce valuable software solutions in a patterned way like other methods do.

The methods of KLUZA & NALEPA [Kluza & Nalepa, 2013] and DECREUS & POELS [Decreus & Poels, 2011], are examples that derive *requirements* from BPMN activities, providing however an unclear relation between business and software, and an incomplete software architecture which is seen as insufficient to be considered as electable for in-house software development. Besides that, the indefiniteness of the relation with the *business process* activities, which may result in many or few user *tasks*, impedes the creation of a stable business and software relation. This increases the probability of generating unnecessary entropy in the software modeling method, producing a solution which may be more complex than the problem itself, i.e. not as much simplified since the beginning as it could, increasing the probability of introducing combinatorial explosions in the system [Aveiro et al., 2015].

It is however possible to identify that by means of providing business semantics, which is the case of the KLUZA & NALEPA method, and doing it by means of identifying business rules tasks (“BR Tasks”), as impositions of the system when dealing with existing BPMN activities, the resulting *requirements* are also semantically similar to *use cases*. The semantics are also applied in the method of PÜTZ & SINZ [Pütz & Sinz, 2010], which presents a more structured derivation of workflow systems, as they derive BPMN models from *business processes* using the Semantic Object Model (SOM) notation, an approach which is close to SOA methods, relating actions and *resources*, and those actions to BPMN activities, however, not designing the User Interface. This situation also happens with the MINERVA method [Delgado et al., 2010], which also relies on SOM and BPMN, in this case orienting the solution to a SOA concerning a final software architecture. A similar approach is presented in the BPLSOA method [Cardona & Duarte, 2013], which applies Software Product Lines (SPL), developing “Service Points” (web services) per BPMN activities.

Figure 4 presents the ZF matching of the BPM methods, which in the most common case find the solution for software generation in business *regulations* and domain concepts, however without contemplating the User Interface design, therefore not being suitable to support the needed service usability and UX demands. When semantics are added to the method, that is reflected in the ZF within the “Scope” level, facilitated by the higher-level concepts used by those methods. The cells which are not specifically covered by BPM methods are related to the “Where” column and “Designer” and “Builder” levels, which is more specifically the human-computer interaction aspects of Information System development, identified as “HCI”.


Business Process Management

	What	How	Where	Who	When	Why	
Planner	Pütz&Sinz Delgado Cardona&Duarte	Pütz&Sinz Delgado Cardona&Duarte		Pütz&Sinz Delgado Cardona&Duarte	Pütz&Sinz Delgado Cardona&Duarte	Pütz&Sinz Delgado Cardona&Duarte	Scope
Owner	BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	Kluza&Nalepa Pütz&Sinz Delgado Cardona&Duarte	Concepts
Designer	Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	Kluza&Nalepa Pütz&Sinz Delgado Cardona&Duarte	HCI	BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	Kluza&Nalepa Pütz&Sinz Delgado Cardona&Duarte	Logic
Builder	Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte		BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	Kluza&Nalepa Pütz&Sinz Delgado Cardona&Duarte	Physics
Imple-menter	Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	Kluza&Nalepa Decreus&Poels	BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	BPMN Kluza&Nalepa Decreus&Poels Pütz&Sinz Delgado Cardona&Duarte	Kluza&Nalepa Pütz&Sinz Delgado Cardona&Duarte	Technology
Operator	THE	ENTERPRISE					Product
	Material	Process	Geometry	Instructions.	Timing	Objectives	

Figure 4. ZF Matching of the BPM methods.

2.3. Human-centric based Methods

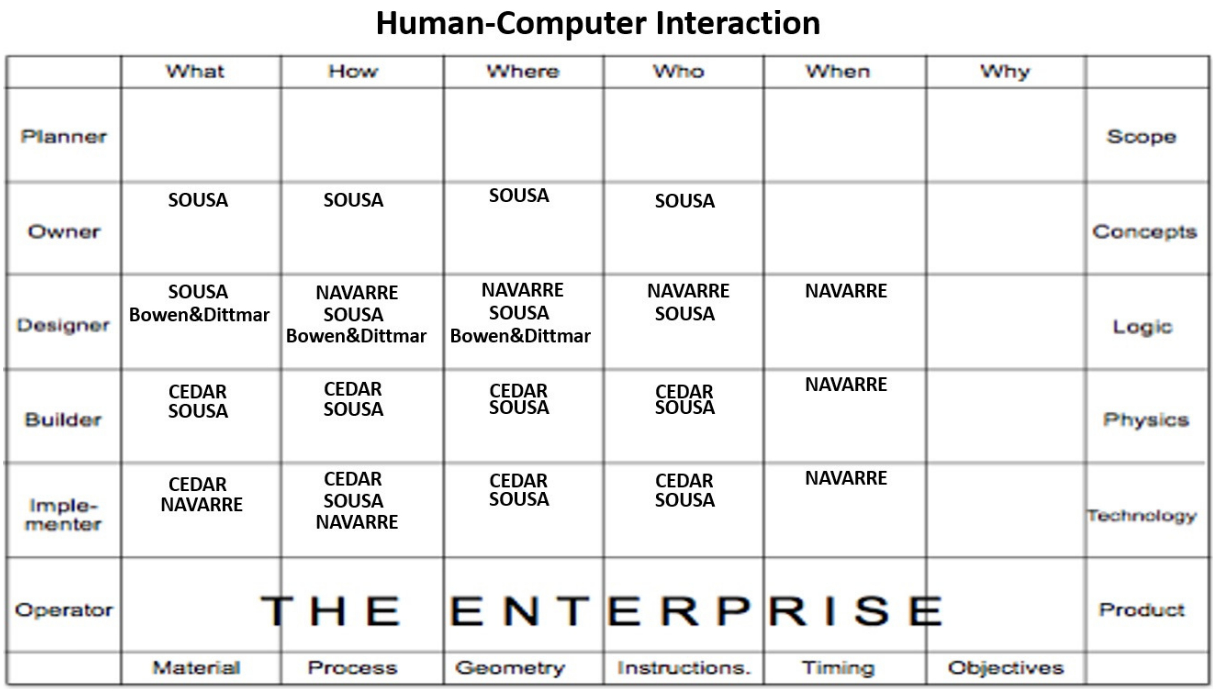
Human-centric methods involve the human (user) perspective in the design of the system, initially bridging HCI and SE [Nunes & Cunha, 1999], by proposing techniques that may be incorporated in the development process [Joshi et al., 2010], which in our opinion should be "native" in the quest for a holistic and effective SDP. In a systems engineering perspective, methods seek to enhance the effectiveness and efficiency of the system grounded on the perspective of the user, instead of (only) the internal structure of the Information System.

2.3.1. Human-Computer Interaction

Human-Computer Interaction (HCI) methods establish architectural relations between the *business process* human activities and the design of the User Interface, yet not always considering the full complexity of the Information System.

Existing methods design the system from a User-Centered Design (UCD) perspective, focusing on the design based on user analysis, from which the CEDAR [Akiki, 2013] method is an example, as it designs the User Interface based on *task* analysis, complemented with *data* models, from which our approach is different as it complementarily conceives the Business Logic layer. The same situation happens with SOUSA [Sousa et al., 2008] method, that structures the software architecture by means of deriving implementation solutions from the decomposition of the *business processes* mapped as "activities" and "tasks"; a straight-forward and practical solution that can be used to generate code of an Information System which User Interface was conceived, based on a user-centered perspective from *business process* models with no traceability gaps, which is a single situation that we find in the literature besides our approach. The SEMI-FORMAL FRAMEWORK [Bowen & Dittmar, 2016] is an example of an HCI method that has concerns related to collaborative User Interface design, since in the HCI domain, the level of elaboration of the User Interface is so detailed that the same User Interface may need working hours from more than one designer.

NAVARRE [Navarre et al., 2009], presents a method that produces enough information to develop a Software Architecture driven by user *task* modeling, and by relating the user actions with system functions, and producing enough semantical information to design a Database. Similarly to agile methods, the business structuring is out of the scope of this method, yet it is compatible with conception following User Task identification, with a User Interface architecture oriented for critical systems, where the interaction level is always specified with the intention of producing immediate system feedback. The feedback is mandatory in an adequate enterprise Information System, yet, this level of feedback is beyond minimal implementation, which is in our perspective is letting the user know the result of its operations.



Human-Computer Interaction

	What	How	Where	Who	When	Why	
Planner							Scope
Owner	SOUSA	SOUSA	SOUSA	SOUSA			Concepts
Designer	SOUSA Bowen&Dittmar	NAVARRE SOUSA Bowen&Dittmar	NAVARRE SOUSA Bowen&Dittmar	NAVARRE SOUSA	NAVARRE		Logic
Builder	CEDAR SOUSA	CEDAR SOUSA	CEDAR SOUSA	CEDAR SOUSA	NAVARRE		Physics
Imple-menter	CEDAR NAVARRE	CEDAR SOUSA NAVARRE	CEDAR SOUSA	CEDAR SOUSA	NAVARRE		Technology
Operator	THE ENTERPRISE						Product
	Material	Process	Geometry	Instructions.	Timing	Objectives	

Figure 5. ZF matching of HCI methods.

Figure 5 presents the ZF matching of the HCI methods, which elicit user *requirements* from human activity and develop a software architecture, covering the greatest part of the software development and SE architectural requirements, with advantages in terms of User Interface design. HCI methods produce software architectures suitable for in-house development, yet, do not usually representative of the business, except the case of the SOUSA method. The relation with the business is structured by means of *task* models and user stories, a relation of user activity and software components targeting user performance. The specific feedback supported by the NAVARRE method is highlighted by filling three layers of the "When" column. The CEDAR method is comprehensive concerning *data* management. The BOWEN&DITTMAR method reveals the level of detail of the HCI methods in terms of User Interface design.

2.4. Software-oriented Modeling Methods

Software models aim at architecting the software parts which are relevant for implementation, in which perspective business and human needs represent the problem under analysis, and the software parts represent the software solution for the Business Process Improvement (BPI). The methods that predict the capture of *requirements* from *business process* models targeting software architecture elaboration, are within the scope of our analysis, to which we provide a historical background of the main field of Software Engineering.

2.4.1. Requirements Engineering

Requirements Engineering (RE) gained importance within the Software Engineering (SE) domain in the 1990's when *requirements* elicitation was established as the first phase of the waterfall model. Later, with the introduction of software development methods, namely the RATIONAL UNIFIED PROCESS (RUP) [Kruchten, 2004], it gained new importance, as the engineering process that would be carried out along the overall SDP, where *use cases* assumed a preponderant role, today extended to other types of *requirements*, namely *goals*. RE methods model the business and its human activity producing specifications that can be used for software modeling. Traceability is a priority expressed by the principle that once *requirements* are elicited, they should be tracked along the lifecycle of the requirement until the generated code, even if the retro-traceability, from the software to the stakeholder strategical needs, is lost in this process over time [Nair et al., 2013]. Traceability efforts mostly concern: pre and post-*requirements* i.e. track *requirements* to their origin, and to their implementation, respectively; and the need to maintain traceability along the requirement lifecycle. Applied techniques mostly rely on *use cases* or *goals* [Loniewski et al., 2010].

KAOS [Lamsweerde, 2000], is a *Goal*-Oriented Requirement Engineering (GORE) that aims at specifying the basic operations that a system must comply with in order to support the business *goals* and the user expectations. KAOS defines a model that relates those *goals* (including the expectations) with the parts of the system which are operated automatically or by human action. The method provides an approach to the specification of the functionality of the system (Business Logic), however, not addressing its full complexity, namely the Database and the User Interface.

The ULLAH & LAI [Ullah & Lai, 2011] method decompose business *goals* and complement them with software models, and are examples of the concern of aligning business and software by means of maintaining traceability, yet not producing a software architecture usable for in-house development. The GONZÁLEZ & DÍAZ [González & Díaz, 2007] method, combines *goals* and *use cases*, and the KORHERR & LIST [Korherr & List, 2006] method withdraws *use cases* from *business processes* modeled using UML Activity Diagrams, yet, both presenting an unclear relation in this extraction (of *use case*s), leaving space for uncertainty in terms of the number of structured *requirements*. The I-STAR (I*) framework [Yu, 1995] considers the complete "tasks" and their dependence relatively to "*resources*" and "*goals*", that may be used to capture the *requirements* for the elaboration of an Information System, which can be used e.g. to model the *data* warehouses [Mazón et al., 2007], solely based on the requirements information produced by I* and the application of software architectural patterns.

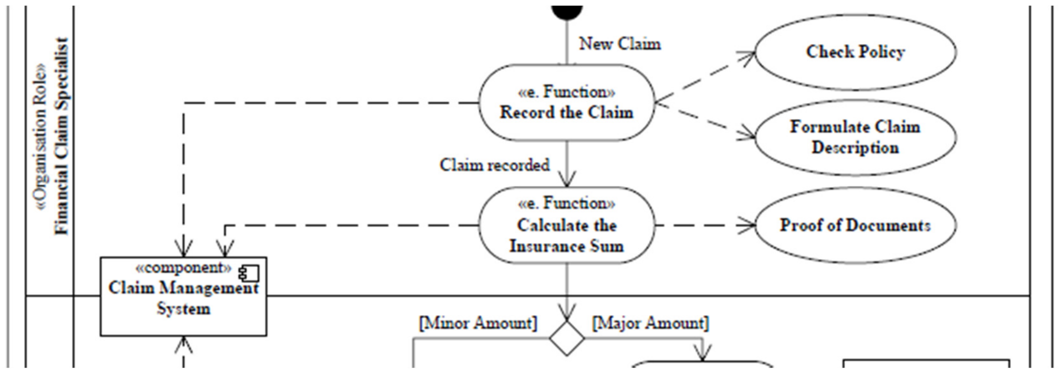
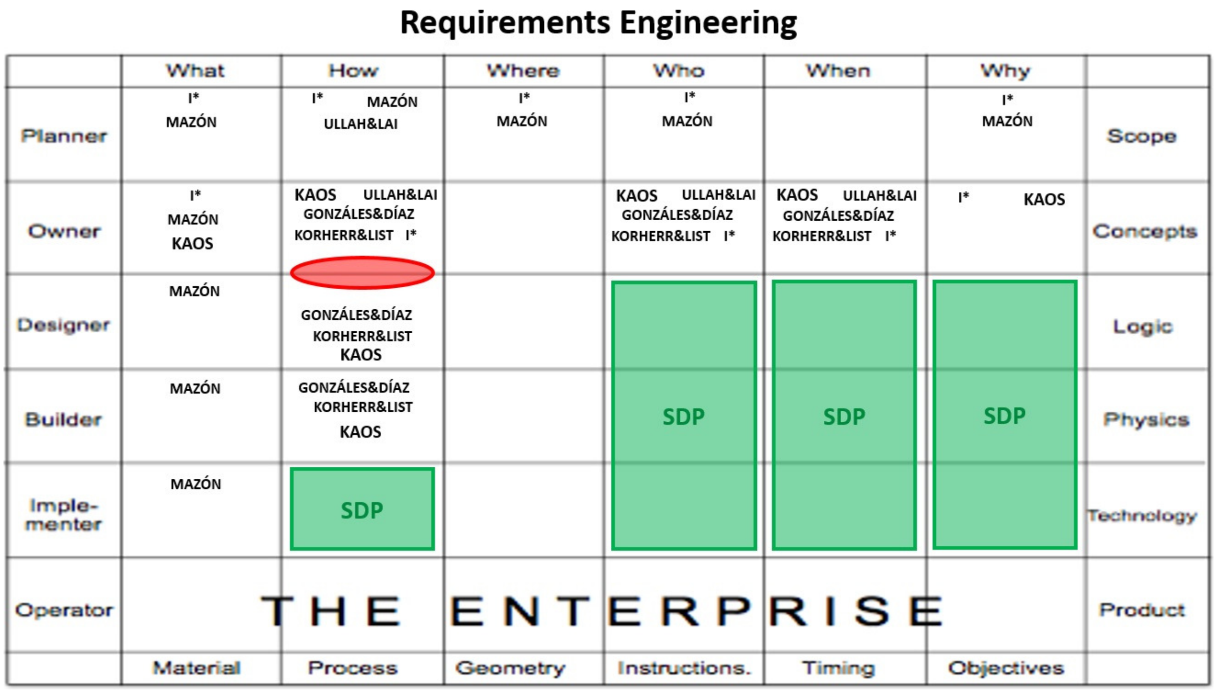


Figure 6. Example of Business Process to Use Case Relation Indefiniteness.

Figure 6 presents one example from the KORHERR & LIST method of how the derivation of *use cases* can be carried out, in this case, from UML activities. The technique method is representative of the problem that we raise concerning the capturing of *requirements* from *business processes*, which we define as a problem of "*requirements* indefiniteness", as it is in most cases impossible to know with how many structured *requirements* (in this case, *use cases*) the user will carry on in his *task*. E.g. the "Record the Claim" and "Calculate Insurance Sum" are performed consecutively, and are related to three *use cases*, and the question that stands is if there is the need for two or three (or more) User Interfaces, or if only one. A similar situation, but based on *goals*, happens in the GONZÁLEZ & DÍAZ method.

Figure 7 presents the matching of the RE methods in the ZF, where it is possible to identify that methods work between the "How" and "What" columns, producing artefacts valid for software development, leaving the rest of the Information System conception to the SE methods that use the elicited *requirements* for their SDP (marked by green rectangles). The *business process requirements* modeling indefiniteness is highlighted with a red oval. I* is the most comprehensive method, namely by modeling the "reasons why" *actors* need to perform their *tasks*, in which is accompanied by the KAOS method. Contrarily to *use cases*, *goal* oriented approaches provide a social perspective that provides context to the design. As such, use cases are more direct to software implementation, and goals are more oriented to the description of the business, including the user.



Requirements Engineering

	What	How	Where	Who	When	Why	
Planner	I* MAZÓN	I* MAZÓN ULLAH&LAI	I* MAZÓN	I* MAZÓN		I* MAZÓN	Scope
Owner	I* MAZÓN KAOS	KAOS ULLAH&LAI GONZÁLES&DÍAZ KORHERR&LIST I*		KAOS ULLAH&LAI GONZÁLES&DÍAZ KORHERR&LIST I*	KAOS ULLAH&LAI GONZÁLES&DÍAZ KORHERR&LIST I*	I* KAOS	Concepts
Designer	MAZÓN	🟥 GONZÁLES&DÍAZ KORHERR&LIST KAOS		🟩	🟩	🟩	Logic
Builder	MAZÓN	GONZÁLES&DÍAZ KORHERR&LIST KAOS		SDP	SDP	SDP	Physics
Imple-menter	MAZÓN	SDP		🟩	🟩	🟩	Technology
Operator	THE	ENTERPRISE					Product
	Material	Process	Geometry	Instructions.	Timing	Objectives	

Figure 7. ZF Matching of the RE methods.

It is based on RE techniques that the existing SDPs rely on in order to be successful in the process of designing software. However, from our analysis we infer that despite targeting software development, methods do not target the support of a complete software architecture, reason why the applied transformations are usually restrict in terms of Information System specifications, an aspect which also explains why traceability of business *requirements* and software implementation traceability is still unclear in terms of a stable patterned relation between business *goals*, *use case*s, and software components specification.

2.4.2. Software Engineering

Software Engineering (SE) began with the challenges related to the management of hardware systems in the 1940's, and was initially referenced as "programming in the large". As systems gradually developed in complexity and required the application of traditional engineering practices, the scientific field acquired the current name of "Software Engineering", leading to the appearance of the first ("Flow") diagrams in the 1940's, and the first high-level languages (ALGOL, FORTRAN, COBOL) in the 1950's. The appearance of high-level languages led to the introduction of theories of "large programming", and software applicability expanded due to his enhanced potential to solve *data* management problems. Operating systems and servers, were more powerful but the problems being solved within enterprises were also more complex, and software entered an era known as "the software Crisis". As a result of this, new methods appeared, and it was after the second NATO conference on SE in 1968, that the software system development lifecycle was firstly divided in five stages: "Specification"; "Design"; "Programming"; "Debugging" and "Maintenance". This led to the application of distinct techniques for each stage in order to gradually control project complexity, and consequently to its definite establishment as an engineering discipline in the late 1980's. New methods and techniques were introduced since then, continuously raising SE development effectiveness, as its current main objective.

Software Engineering (SE) methods are fairly complete approaches regarding business and software modeling, and may be divided in architectural-centric and agile. Architectural-centric methods focus in attaining certain quality attributes of the software architecture under development, and usually use OBJECT-ORIENTED SOFTWARE ENGINEERING (OOSE) [Jacobson, 1992], in order to apply *use case*-driven techniques and derive a software architecture from business models and *requirements*. *Use cases* were integrated in the RATIONAL UNIFIED PROCESS (RUP), a refinement of the OOSE-based UNIFIED SOFTWARE DEVELOPMENT PROCESS [Jacobson et al., 1999], where one *use case* represents at least one user interaction with the system with a certain purpose, eliciting *use cases* based on the design of *business processes* using UML Activity Diagrams, and developing the architecture of the software based on a *class* Domain Model [Booch et al., 2005].

In this process, the relation between *use cases* and the *business process* human activities is not structured in a clear way, as the *use case* scope is left to the analyst decision, leading to a variance in the number of *use cases* that may support one specific *task* under the user responsibility. This situation, earlier described as *use cases* "indefiniteness", leads us to a situation where the "What" and "How" columns of the ZF become inconsistent. This happens because the relation between both can be derived in different ways, and with different results. As examples, this situation happens in the methods of GRAHAM [Graham, 1996], BIDDLE [Biddle et al., 2002] (despite using *essential use cases*), the FORMAL DESIGN ANALYSIS FRAMEWORK (FDAF) [Dai & Cooper, 2007], and DENNIS [Dennis et al., 2012]. The situation is further exemplified in the analysis of the *business process* (left) and *use case* (right) meta-model of the FDAF method presented in Figure 8, in which it is not possible to identify common structures between both. The *business process* is observed in the processing perspective, and the *use case* decomposition considers the *actor*, which however does not exist in the business meta-model.

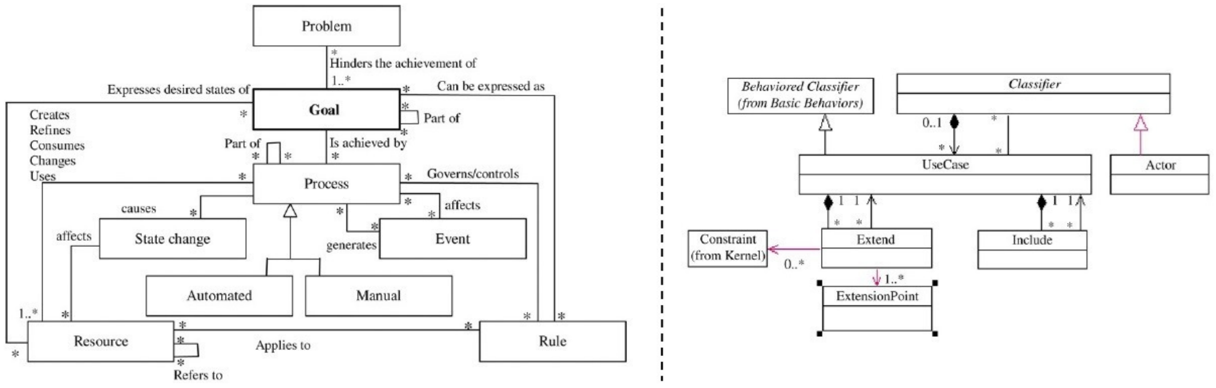


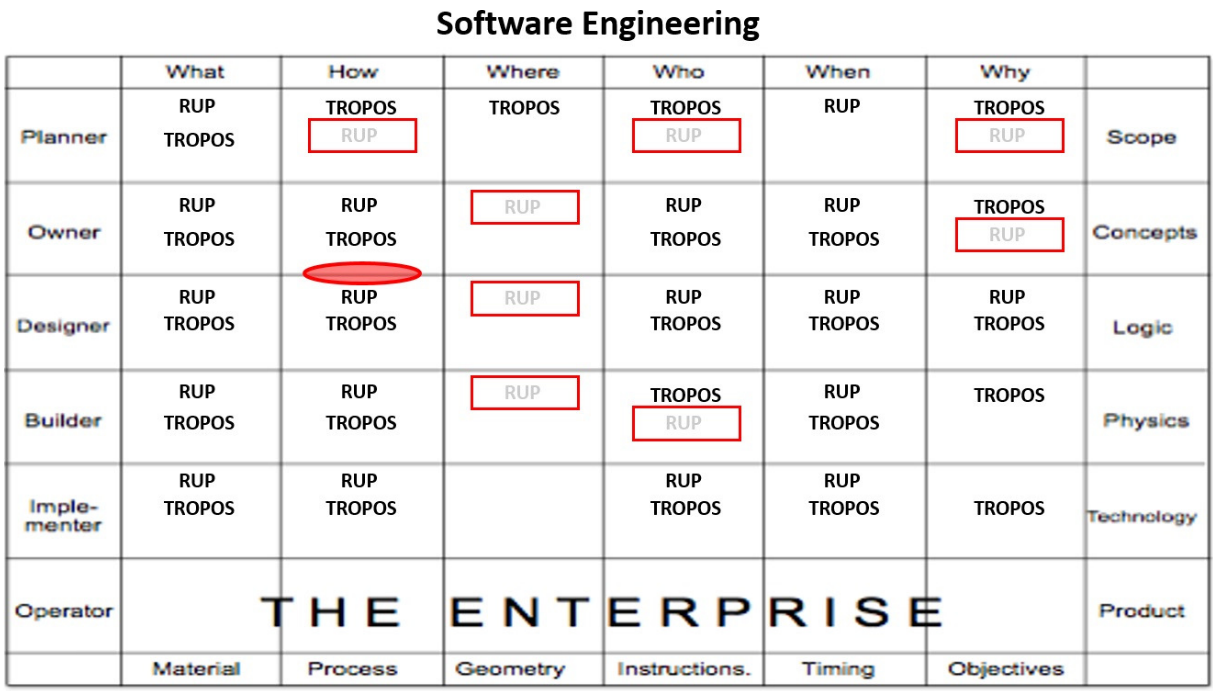
Figure 8. FDAF Meta-Model of Business Process and Use Cases.

TROPOS [Morandini et al., 2014] represents a distinct SE perspective, as a method which is oriented to the specification of agents and the capabilities that an Information System must address in order to fulfil the business model. TROPOS bases the specification of the business model on i*, from which information it implicitly withdraws the functions that implement each capability by means of the specification of Activity Diagrams [Bresciani et al., 2004], in order to apply the OO Analysis & Design engineering process in order to elaborate the architecture of the system.

TROPOS is oriented to the interaction between agents, and for that purpose considers the *goals* and *softgoals* of the agents, an information that includes emotional expectations, which once combined may induce the specification of complementary functionality of the system. The interaction between *actors* is specified including the performed *tasks* and intentions. However, business *regulations* and the User Interface design is not contemplated. Moreover, the relation from the specification of *goals* and capabilities allows distinct specifications for the same problem, depending on the implementation strategy, and therefore, indefiniteness is also introduced in this case, making TROPOS closer, but with improvements, when compared to RUP and OOSE specification.

Villiers elaborated the matching of RATIONAL UNIFIED PROCESS (RUP) with the Zachman Framework (ZF) in order to assess the method [Villiers, 2003], and proceeded by matching the artefacts produced by RUP in each of cell, applying the extension of the ZF for Information Systems architecture (Figure 9), to which we have the following observations:

- Unstructured Descriptions: In the matching, it is possible to identify that in the “Scope” view, the *requirements* are matched only by an unstructured description, and for this reason cannot be validated as artefacts that ensure traceability between business and software modeling. The same applies to business rules.
- Use Cases Scope: The usage of *use case*s (Function and Owner’s View cell) to detail the *task* of the *actors* (“People” and “Owner” View cell) is not sufficient to represent the business activity, namely its *business processes*, leading to uncertainty regarding how many *use case*s represent a complete *task* of the user, therefore leaving this relation specification to the business and software modelers decision.
- “Network” as a “Where” representation solution: We do not agree on the physical *data* network, *nodes* and *file* location, as a specification of the “Where” column, since we believe that the network should only be represented at the Operational (or Product) level, and oppositely, the place where things may happen, is being omitted.



Software Engineering

	What	How	Where	Who	When	Why	
Planner	RUP TROPOS	TROPOS RUP	TROPOS	TROPOS RUP	RUP	TROPOS RUP	Scope
Owner	RUP TROPOS	RUP TROPOS	RUP	RUP TROPOS	RUP TROPOS	TROPOS RUP	Concepts
Designer	RUP TROPOS	RUP TROPOS	RUP	RUP TROPOS	RUP TROPOS	RUP TROPOS	Logic
Builder	RUP TROPOS	RUP TROPOS	RUP	TROPOS RUP	RUP TROPOS	TROPOS	Physics
Imple-menter	RUP TROPOS	RUP TROPOS		RUP TROPOS	RUP TROPOS	TROPOS	Technology
Operator	THE ENTERPRISE						Product
	Material	Process	Geometry	Instructions.	Timing	Objectives	

Figure 9. ZF Matching of the SE methods RUP and TROPOS.

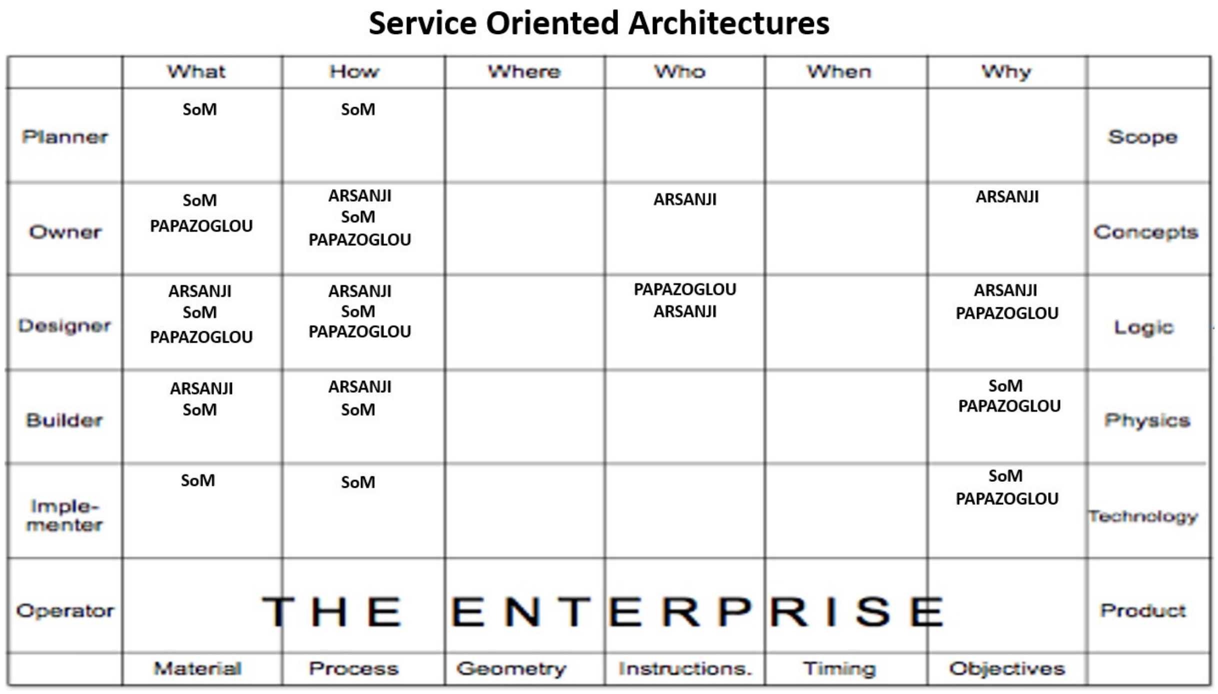
The ZF matching of RUP (by Villiers) and TROPOS is presented in Figure 9. RUP based covers the greatest part of the software modeling destined ZF cells, below the "Owner" View and including it, there is an unclear relation of *use case*s with the *business process* modeling, which can be seen as the space where imprecision occurs in Information Systems development within enterprises (represented with a red filled square). Moreover, more abstract concepts, namely from the "Scope" view, are not considered in a structured way, and are produced by narratives, and therefore do not ensure full business to software traceability (represented with a red oval). TROPOS, by means of the usage of I*, covers "Scope" layer cells, and further fills the "Why" column cells by means of the specification of the reasons why *actors* perform certain *tasks* based on *softgoals* modeling. TROPOS does not cover the User Interface design, however, it creates conditions for the application of methods that may cover this chasm, as the indefiniteness of *requirements* is reduced by the usage of the I* practice of specifying a single *task* (the "dependum") between two *actors* (the "depending" and the "dependee").

Agile Methods (AM) target the automation of the system based on the incremental and gradual of software for the support of the business activity. They usually rely on e.g. EXTREME PROGRAMMING (XP) [K Beck, 1999], SCRUM [Schwaber, 2004] or SAFE [Scaled Agile, 2017], to be operated, which are out of the scope of analysis, as they are SDPs which may be integrated with any object-oriented modeling language to implement BPIs at the user *task* level. Moreover, the development of architectural spikes is not observed as a positive approach to enterprise Information Systems development, because of the need to avoid refactoring and maintain business traceability. The same applies to Software Process Improvements (SPI), which intent is to control and improve the SDP [Development, 2010; ISO, 2003; Martins & da Silva, 2016], which are usually also oriented to small and medium enterprises [Laporte et al., 2008; Paternoster et al., 2014], matching our future work namely concerning effort estimation and project management in the quest of enhancing the ROI of software development. Thus, for a purpose of disambiguation, GOALS is to be compared to business and software development languages and methods and not SDPs and SPIs, considering the problem being solved.

2.4.3. Service Oriented Architecture

The SD domain methods that provide a software development support, mostly concern service interoperability business-to-business as implemented by Service Oriented Architecture (SOA) methods, which usually deploy web services settled in object-oriented software architectures. SOA is however not oriented for User Interface design.

The approach is based in *business process* modeling for the specification of clusters of web services that are able to send and receive *data*, and perform operations over this *data* on user demand (by means of e.g. a third party developed User Interface). In SOA, the user focus is on the list of services which are provided, which are extracted from *business processes* [Arsanjani et al., 2008; Bell, 2008; Erl, 2005], using *use cases* SOUP [Mittal, 2006], or doing it independently of the used technique [Papazoglou et al., 2006].



Service Oriented Architectures

	What	How	Where	Who	When	Why	
Planner	SoM	SoM					Scope
Owner	SoM PAPAZOGLOU	ARSANJI SoM PAPAZOGLOU		ARSANJI		ARSANJI	Concepts
Designer	ARSANJI SoM PAPAZOGLOU	ARSANJI SoM PAPAZOGLOU		PAPAZOGLOU ARSANJI		ARSANJI PAPAZOGLOU	Logic
Builder	ARSANJI SoM	ARSANJI SoM				SoM PAPAZOGLOU	Physics
Imple- menter	SoM	SoM				SoM PAPAZOGLOU	Technology
Operator	THE	ENT	ERP	RIS	E		Product
	Material	Process	Geometry	Instructions.	Timing	Objectives	

Figure 10. ZF Matching of SOA methods.

Figure 10 presents the matching of the SOA methods, which cover the "What" and "How" columns, which are the ones which are more closely related to the *data* and function layers of Information Systems concerning the MVC model, also known as Database and Business Logic. In the cases that *use cases* are used for functional *requirements* elicitation, the first layer of the ZF is also covered by SOA methods. The "Where" and "When" columns are not covered by SOA methods, as those articulations do not assume an important role beyond architectural service invocation precedence imposition

Despite the fact that the User Interface is not designed in these methods, they are relevant in terms of the business and software modeling scope and can eventually be completed with other methods in order to produce a User Interface design, in the process of increasing the enterprise software development success rate. An approach that may produce similar results is the SOA derivation methods which are based on a user-centered perspective, to which the traditional SOA methods can be applied concerning Information System implementation.

2.5. Matching of the Goals Approach

The matching of the GOALS Approach with the ZF, serves the purpose of understanding the scope of the language and method, and provides an overview of the concepts, including their stereotypes, which will be presented in the next Chapters. GOALS covers 24 of the 30 cells of the ZF. They refer to: the organization of business in the "Scope" layer, of the human activity in the "How" column, of interaction spaces in the "Where" column, of the user in the "Who" column, of relations in the "When" column, and of reasons in the "Why" column. GOALS provides a notation which are composed by the stereotypes which are presented in each of ZF cells in Figure 11. The planner cells are filled with a semiotic logic that gives meaning to each column, and which are considered separately from the 24 covered cells, as they are undergoing future work, suiting however the purpose of explaining the remaining concepts.

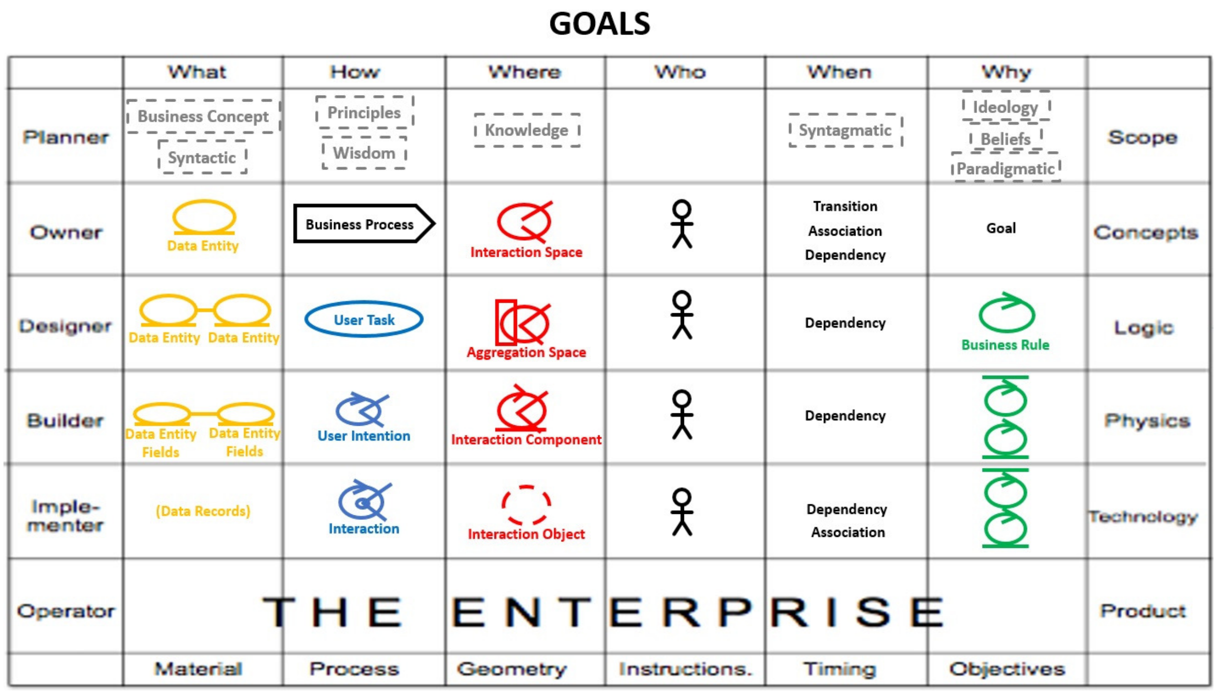


Figure 11. ZF Matching of GOALS.

In Figure 11, the "What" column, the *business concept* represents the information related by means of *syntactic* relations. In the "How" columns, the human activity is described in terms of User Tasks, User Intentions and User Interactions, which in the semiotic logic, should be applied according certain *principles* and *wisdom*, based on the *knowledge* which is provided by the components of the "Where" column, reflecting the information presented by the system. In the "Who" column, the *actor* may be modeled at different levels, namely using the information from his *tasks*, intentions and interactions. We leave the cell empty at the "Planner" level because the system can be used by anyone, and for that reason leave that specification void. In the "When" column, the *syntagmatic* meaning refers to the orientation to keep historical *data*, in the "Concepts" level, the *transition* specifies the *business process* design, and the *dependency* relation is used for the remaining relation between concepts.

Hence, the GOALS business and software models allow covering the ZF from the "Concepts" to the "Technology" layer, which is where our focus is, as they are the ones which are sufficient to bridge EE and SE by means of HCI. In this delimited zone of the ZF, GOALS proposes a solution to solve a critical issue of User Interface and system conception by means of the introduction of the interaction space concept in the business representation, as the space where users carry on their *tasks*, suggesting the implementation of the interaction between users and between users and Information System in a single User Interface (by default) with the objective of diminishing human effort by means of reducing navigation [Winckler et al., 2015]. This is possible because it is likely that most of the User Interfaces may be implemented in a single screen, without the need to alternate with other system interfaces, a mechanism which however may be present when needed.

2.6. Traceability

The analysis of Software Product Lines (SPL) facilitates understanding how GOALS solves the business and software traceability problem, despite the fact that the elaboration of SPLs is out of the scope of the problem being solved, which refers to a single system development. We use the example and problem statement provided by the CONCEPTUAL VARIABILITY MODEL (CVM) [Berg et al., 2005], to illustrate the problem of traceability as observed by SPLs.

In the top of Figure 12, the phases of a SDP are divided in the "Problem" and "Solution" spaces, and at the bottom, the traceability problem refers the need to trace multiple "Requirements" (from the "Problem Space"), to the "Design" elements which will be implemented (in the "Solution Space"). It references a many-to-many relation between both sides, which may become unmanageable because of the need to track the changes between multiple distinct models. The described traceability problem is "shortcut" by GOALS by means of representing the "Problem Space" in the "Solution Space" (as highlighted in the top of the Figure). I.e. the representation of "Requirements" is materialized as "Design Elements" that will be implemented (programmed) as a structural part (the "back-bone") of the software solution. Thus, the GOALS Software Architecture represents both the "Problem" and the "Design".

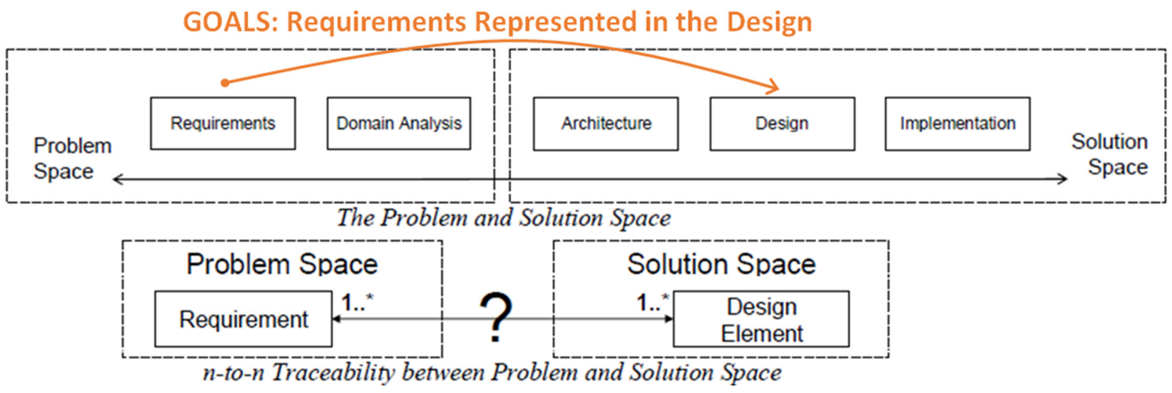


Figure 12. CVM Traceability Problem Specification.

This applies to the phases of "Requirements", "Domain Analysis", "Architecture" and "Design" considering the CVM. Therefore, traceability is not lost as long as the business and software models are updated. This means that as long as traceability of design objects to code is also not lost, then traceability between the business model and software implementation is maintained. As GOALS is independent from implementation, the traceability from the design model to implementation is dependent on the software development framework and the relation that is established to the design for each type of component. As such, if the relation between the Software Architecture and the each implemented class is of one-to-one, traceability is trivial, if not, patterns to maintain traceability, automatically if possible [Tryggeseth & Nytro, 1997], should be considered, yet only between the design model and the source code implementation.

Comparing the CVM and GOALS, the identification of the commonalities ("Problem" and "Solution" common parts) and variability ("Problem" and "Solution" distinct parts) of the lines of software production, may also be carried out differently. Since with GOALS the "Problem" is represented in the "Design", the identification of the differences may be carried out by means of identifying the common and distinct parts using a single model (the Software Architecture), with no need to "track-back" the "Design" to the "Requirements". This facilitates understanding which version of the *requirements* and which version of the remaining Software Architecture components is valid in different SPLs.

2.7. Conclusions

The matching of the identified method in terms of filled cells in the ZF targeting traceable architectures between business and software is presented in Table 2. The SD, EE and BPM methods fill 19, 25 and 27 cells respectively, reflecting the representation of the business, and the generation of workflow and organizational operative systems from the design of *business processes* (in the cases of BPM and EE). The RE field, fills 14 cells of the ZF, which refer to the purpose of the domain, which is the establishment of the relation between business and software, and not the targeted software development. The HCI and SOA domains fill 19 and 16 of the cells, reflecting the produced software architecture, which are User Interfaces without specified Database design (in HCI), and Database and Business Logic without User Interfaces. The SE domain, namely RUP and derived methods, fill 26 of the cells, yet, not covering the higher levels of specification. GOALS fills 24 cells with which it models the business and software, plus five cells of the "Planner" layer which are used to support the specification of the remaining concepts in this thesis. However, as the semiotic layer is part of ongoing work, these cells are still not considered in the final count.

Table 2. Results of the ZF Matching.

Comparison	**Matched Cells**
Service Design (SD)	19
Enterprise Engineering (EE)	25
Business Process Management (BPM)	27
Human-Computer Interaction (HCI)	19
Requirements Engineering (RE)	14
Software Engineering (SE)	26
Service Oriented Architecture (SOA)	16
GOALS	24

The importance of the number of filled cells is relative, and is merely indicative of the coverage which results in each domain in order to fulfil existing challenges, in this case in the perspective of business and software traceability. Nevertheless, considering the holistic perspective of the enterprise provided by the ZF, which considers the business and software models as complementary parts, the coverage of the 30 cells should be prospected when aiming a complete enterprise Information System modeling and code generation.

In a qualitative analysis, the techniques provided by RE methods are crucial for the relation of business and software models, and we believe that by means of further specifying the relation between *goals* and *tasks*, including user-centered concerns, more effective traceable standards may appear. The SD challenges can be considered as an ultimate target concerning usability when observed from the software modeling perspective, since the derived *requirements* are more representative of the business than those of any other domain, which once fully represented and structurally related to a User Interface structure, may represent an added value in systems development. It is precisely in this gap, the User Interface interaction modeling of human and system, the main purpose of the HCI domain, and an absolute and still uncovered need in SE, that we believe that is the weakest link, and also the key, to cover of the gap between the EE and the SE domains. SOA are the best example of how software development may become productive if it is possible to work with one less tier of software architecture, the User Interface, considering an MVC model. SOA is not sufficient in order to cope with the demands of an enterprise Information System, yet, they are an example of how traceability may be straightforward if it is settled in firm and simple relations between distinct concepts, even if one is originated from business (the *business process* and/or the service) and another from software modeling (the web service).

The main distinction between GOALS and the analyzed methods happens in terms of the number of *use cases* which are identified in the SDP as derived from *business processes*, and in the way that those *use cases* are further decomposed into intentions and interactions, removing uncertainty from the business-to-software modeling traceability, and solving the *requirements* "indefiniteness" problem, as will be presented throughout the thesis. The pattern-based GOALS traceability structure is complemented with the representation of the business model in the software model, which ensures that traceability between the enterprise relevant conceptual constructs is not lost. The approach allows designing the user *task* with detail and establish a direct relation with the software architecture, instead of extending the originally identified *use cases*, and losing track of the type of human behavior (intention, interaction, etc.) and its supportive software implementation.

The GOALS process, structure (language and notation), method and techniques are presented in the following chapters.

3. The Goals Language

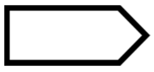
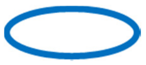
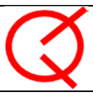
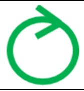
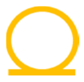
The GOALS modeling language is composed by concepts that model the business and the software of the enterprise. These concepts are related in a single structure, and for each, there is a definition, a symbol, and a logic relation with the other concepts. Each concept is expressed by a Unified Modeling Language (UML) class which stereotype is the concept' symbol, and the structure is a relation of usage between the classes with a multiplicity of many-to-many. Each of the concepts and its meaning within the language structure will be presented as follows.

3.1. Structure

In order to develop the GOALS language structure, two architectures have been defined to represent the enterprise: the business and software architectures, the Enterprise Structure and the Software Architecture, respectively. Together, they compose the Enterprise Architecture, which is complemented with a process, a method and a set of techniques, which represent the guidelines and way-of-building of business and software models. The Enterprise Architecture concepts provide the notation by means of UML class stereotypes (the symbol), and the relation between the classes is expressed by dependency (usage), meaning that one component depends on the good functioning of the other in order to properly work.

The Enterprise Structure describes the business, which briefly states that: a Business Process (BP) is composed of User Tasks (UT), which are performed by users (or *actors*) in Interaction Spaces (IX) (whether physical or electronic), applying the enterprise' Business Rules (BR) over defined Data Entities (DE, which are *business concepts*). The concepts, definition, origin and symbols of the Enterprise Structure are presented in Table 3.

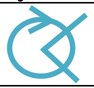
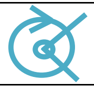
Table 3. Enterprise Structure Concepts Definition, Origin and Symbol.

Concept	Definition	Origin	Symbol
Business Process (BP)	*A Network of UTs that Lead to a Goal*	GOALS	
User Task (UT)	*A Complete User Task within a BP*	AM	
Interaction Space (IX)	*The Space that provides UT Support*	WISDOM	
Business Rule (BR)	*A Business Restriction over DE's Structural Relations*	DEMO	
Data Entity (DE)	*Persistent Information about a Business Concept*	WISDOM	

The Enterprise Structure allows the modeling of the business human activity, of the available *resources* (physical or logical) and the existing *goals*. Maintaining the same top-down logic, the modeling towards the Software Architecture is carried out by means of the User Interface which supports each UT. The "bridge" between the Enterprise Structure human activity and the Software Architecture is achieved by means of Human-Computer Interaction (HCI) techniques, establishing a relation of UTs, User Intentions and User Interactions with the User Interface' Aggregation Spaces, Interaction Components and Interaction Objects, respectively. The relation maximizes the modeling of human activity in terms of meaning, and minimizes it in terms of number, inducing the simplification of the system back from the business design.

The interaction concepts which complement the UT in terms of human behavior structuring, are presented in Table 4.

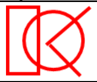
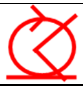
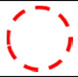
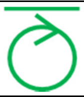
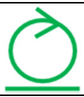
Table 4. Interaction Design Concepts Definition, Origin and Symbol.

Concept	Definition	Origin	Symbol
User Intention	*The Sequence of Actions that the User Wishes to Perform to Complete a UT*	AM	
User Interaction	*The Interaction Inputs of the Information System Performed by the User*	GOALS	

The Software Architecture defines that an: Aggregation Space (AS) is composed of Interaction Components (IC) that trigger User Interface System Responsibilities (UISR) and Database System Responsibilities (DBSR, both belong the Business Logic layer) by means of the Interaction Objects (IO) and remaining User Interface structure (AS and IC) where HCI occurs (UT, User Intention, and User Interaction), in order to manage existing Enterprise Structure *data* (Data Entities, DE), according to the enterprise *regulations* (Business Rules, BR), as ensured by the Interaction Space (IX).

The AS, IC, IO, UISR and DBSR are the five software-specific concepts of the Software Architecture, which are architecturally related to the Enterprise Structure by means of the specific relation of one-to-one between the (software) AS and the (business) UT, and of the UT with its supportive IX, inheriting its BRs as parts of the Business Logic, and the DEs as part of the Database. The software-specific concepts, definition, origin and symbol of the Software Architecture are presented in Table 5.

Table 5. Software Architecture Specific Components, Definition, Origin and Symbol.

Concept	Definition	Origin	Symbol
Aggregation Space	*A User Interface*	HYDRA	
Interaction Component	*Tool of a User Interface*	WISDOM HYDRA	
Interaction Object	*A User Interface Object that triggers SRs*	HYDRA GOALS	
User Interface SR	*A SR that provides support for User Interface data presentation*	AM/GOALS	
Database SR	*A SR that manages data*	AM/GOALS	

The concepts and their symbol provide the notation of the language, and the structure that relates both business and software concepts provides the syntax. The semantics of each concept is provided by a semiotic logic (the Ideology) that is applied to the Enterprise Structure, the Software Architecture and the Human Activity, in order to provide an understanding concerning the meaning of each concept and relation of concepts. The semantical specification is direct to the Enterprise Structure and reflected in the Software Architecture by means of a "meaning" to the IX, BR and DE concepts that remains valid to the User Interface, Business Logic and Database layers. The Ideology, Enterprise Structure and Software Architecture concepts relations are presented in Figure 13.

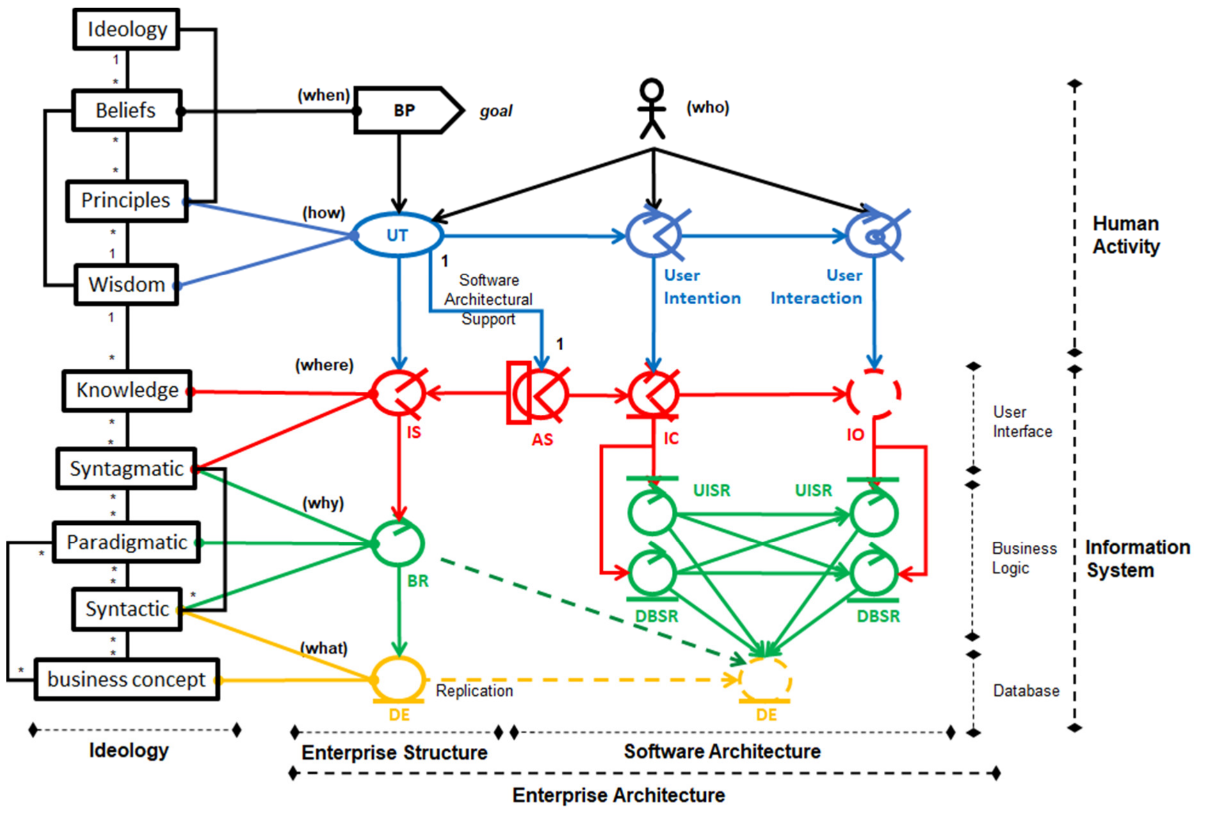


Figure 13. The GOALS Language Structure.

Figure 13 presents the Enterprise Structure and Software Architecture concepts meta-model (considering the M2 level of the OMG modeling infrastructure), using UML classes which stereotype is the concept's symbol, and the dependency relation is represented using a filled line (instead of a dashed line), for the purpose of meta-model interpretation. All the relations are of many-to-many, except for the one-to-one (1 to 1) relation of UT and AS. The relation of UT and AS is defined as of one-to-one (identified as *Software Architectural Support*) in order to promote business and software organization and traceability, establishing that when an *actor* performs a UT he does it by using a single AS, which implementation must provide all the desired system behavior. The *Software Architectural Support* is further detailed in Sections: 3.2.2. User Task (UT), 3.3.1. Aggregation Space (AS), 3.3.2. Interaction Component (IC), and 3.3.3. Interaction Object (IO). The remaining concepts are presented in the remaining sections.

3.1.1. Semantics

The semantics of the language is provided by means of the Ideology (structure of semiotic concepts) presented in Figure 13, which states that: an enterprise is an organization that has a certain *ideology* based on its purpose for the surrounding society, considering or not the related cultural premises. The *ideology* is based on certain *beliefs* and *principles*, which should drive the enterprise towards achieving success (its *goals*). These *principles* should drive the *wisdom* which users apply when carrying on their *tasks*, as they are materializing the enterprise's *beliefs*, and should also be based on existing *knowledge*, which should be structured by means of *syntagmatic* relations, relations which are then composed of other *paradigmatic* and *syntactic* relations of *business concepts*. *Paradigmatic* relations are (complex) relations which are based on *syntactic* relations between *business concepts* of the enterprise. The *ideology* targets project team focus. The *ideology* and the GOALS conceptual approach is further presented in Appendix A – The Goals Conceptual Approach.

3.2. Enterprise Structure Concepts

The Enterprise Structure is a relation of concepts that represents the business of the enterprise. The concepts are, from top-to-bottom: The Business Process (BP), the User Task (UT), the Interaction Space (IX), the Business Rule (BR) and Data Entity (DE). It provides a structured view of the concepts which are relevant for the functioning of the business and how they are related with each other. It allows business management by means of providing a single view of its structure, targeting its validation before software development.

The Enterprise Structure is also the first step aiming enhanced cooperation between business and software modelers, as its concepts may be recognized by both domains with the inclusion of the IX as the common (hybrid) linking concept between the user responsibility (the UT) and existing *regulations* (BRs), which are common concepts to the business and software functioning, and which are based on the existing structure of DEs (the Domain Model). The IX represents the physical and the logical space that is used by *actors* to perform their *tasks*. Being a hybrid (business and software) concept, it can be used to model the physical space (e.g. a room), and also the logical space which will be replaced by an Aggregation Space (a User Interface), for the support of an *actor* perspective of the IX when interacting with the enterprise by means of the Information System.

Each of the Enterprise Structure concepts are presented in the following sections, explaining its origin, adaptions to the original definition and structural relations to other GOALS concepts.

3.2.1. Business Process (BP)

Origin

The Business Process (BP) concept is originated from software development and *business process* organization practice, to promote the dialogue between *Business Owner*, *Business Process Manager*, and *Software Architect* in order to organize the User Tasks (UT) of the *actors* involved in a BP Improvement (BPI). From the modeling of BPs using the ERIKSSON-PENCKER method [Eriksson & Penker, 2000], emerged PROCESS USE CASES method [Valente & Sampaio, 2007], that enabled the documentation of a BP using *essential use cases* for the representation of complete UTs. The replication of the process allowed the stabilization of the definition of the used concepts, namely: the BP and its *goal*, the *business concepts* (e.g. person, degree, etc.), and the complete UT, which were perceptible by business and software modelers, but especially by the users ("*Person or Group of Persons Whom Uses the System*").

Definition

The coupled definitions of BP and *goal* are presented as follows:

- Business Process (BP): *"A Network of User Tasks that lead to a Goal"*. Each BP is a sequence of (at least one) UTs, which are interrelated in a given Business Logic, leading to the achievement of a *goal*. Once the BP is successfully completed, a new objective of the enterprise has been accomplished. The definition of the *tasks* that compose each BP (User Tasks) is presented in the following section.
- *Goal*: "*The objective of the BP*". A *goal* is defined as the objective of the BP, names the BP, and is always the *act* of achieving a *business concept* (defined as a *concept* that has meaning for those who have knowledge about the enterprise), and is represented by means of one or more Data Entities (DE). Any *business concept* can be a combination of any other *concepts*, as defined in semiotics [Damjanovic et al., 2005].

Meta-Model and Example

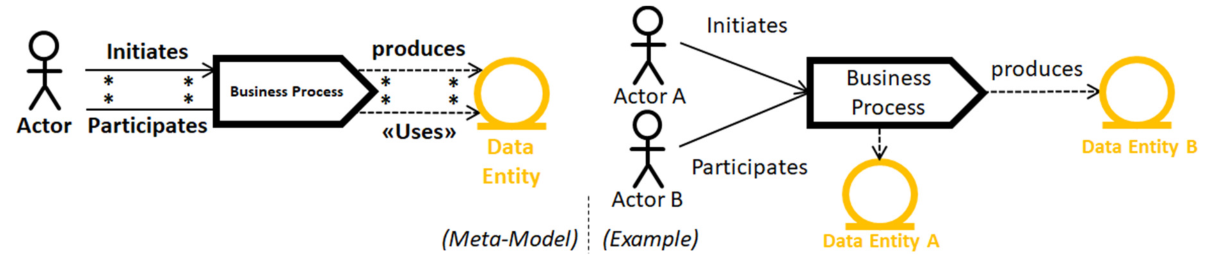


Figure 14. Meta-Model and Example of Business Process (BP).

The meta-model of the BP is presented in Figure 14, and specifies that many *actors* can "Initiate" a BP, and many *actors* can "Participate" in it, and also, that many Data Entities (DE) can be used and produced by a BP. The example presents a situation where "Actor A" initiates and "Actor B" participates in the "Business Process" BP, which uses "Data Entity A" and produces "Data Entity B".

Semiotic Logic

The BP implements *beliefs* by recurrently achieving *goals* that support the enterprise *ideology* within the society. These *beliefs* apply certain *principles*, from which may be generated *wisdom* from the existing *knowledge*. For example, if our enterprise system is a public university, the "solidarity" *principle* may be present e.g. by means of the *belief* that "by facilitating access to studies to more persons, the purpose of raising the society's education level will be better fulfilled" may exist.

Meaning to the SDP

The BP is usually settled in a unique set of related enterprise *business concepts* (DEs) which support the BP execution, and compose the enterprise Domain Model (as will be presented in Sections 3.2.5. Data Entity (DE) and 4.3.2.3. Step 7 – Database Structuring). The relation between the BP and the set of DEs increases awareness on the problem begin solved, namely for *Service Designers* and *Business Process Managers*, by means of deepening their knowledge on the specific part of the enterprise being evolved. This facilitates the BPI development, resulting in faster and more productive meetings, increasing the probability of developing projects in less time. Moreover, the relation between BPs, *actors* and DEs, provides a view of what each BP uses and produces as DEs (products and services) in order to achieve its *goal*, reflecting the enterprise production logic from a higher level.

3.2.2. User Task (UT)

Origin

The User Task (UT) is originated by the need to analyze Business Processes (BP) so that the *Software Architect* may identify who would make what, so that every needed information is processed according to each *actor* (a person or group of persons) responsibility. It was noticed that by means of the application of the WISDOM method there was a clear tendency to identify the steps that the user had to carry on in the User Interface in a single Activity Diagram [Booch et al., 2005], reflecting what the user "would like" to do (User Intentions) when carrying on his *task*, so that he would finish it as quick and accurately as possible. This approach represented a change of logic when compared to "traditional" *use cases*, as it induces simplification by withdrawing unnecessary steps, so that user performance may be raised, and the number of implementation parts may be minimized while maintaining (or raising) effectiveness, therefore, (probably) increasing the efficiency and performance of the system.

This focus led to the implementation of User Interfaces that allowed the execution of every User Intention in the same screen, without navigation, in almost all cases. Thus, by focusing on what the user needed, instead of the existing *data* concepts and business *regulations*, it was possible to promote the simplification of the system, which is reflected in less man-hour effort, promoting the enterprise (business and software) development performance by promoting the user performance.

Definition

The User Task (UT) definition comes from the definition introduced by Larry Constantine and Lucy Lockwood of *essential use case* [Constantine & Lockwood, 2001], as a "*Complete and Meaningful Task Carried Out in Relationship with a System*", and as an evolution of the "traditional" *use case* introduced by UML [Booch et al., 2005]. This brought a relevant change when applied to BPs, since: if two *use cases* of the same person were consecutive, then they would be the same; once the *task* will only be totally completed when the second *use case* is finished. This is a realistic and pragmatic perspective of the UT within the context of a BP, since it is not necessary to represent the *task*'s intermediary steps in the BP design. In an extreme case of importance in which their representation would be needed, then their *task* description could be concatenated (e.g. using a "+" sign) in the *essential use case* (UT) name, still representing the full *task* in the same way.

Meta-Model and Example

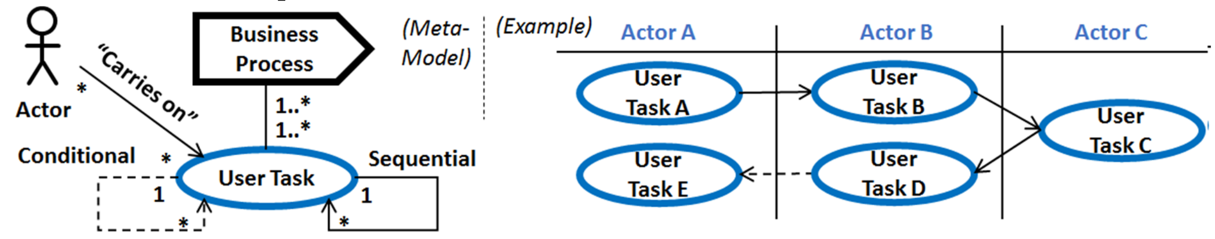


Figure 15. Meta-Model and Example of User Task (UT).

Figure 15 presents the meta-model of the UT, which defines that one *actor* may carry on many UTs and that a UT can also be carried out by many *actors* (as cooperative work); one BP can have many UTs; one UT can belong to many BPs; and UTs are related consecutively or conditionally. The example presents a BP where the flow is defined by means of an UML Activity Diagram (in which the initial and final points are omitted), that identifies 5 UTs which are performed consecutively from *actors* "A" to "B", "B" to "C" and "C" to "B", and eventually to "Actor A" that may execute it or not (as the relation of the "User Task D" with the "User Task E" is conditional).

Semiotic Structure

The user will use the *knowledge* (*syntagmatic* and *paradigmatic* relations) provided by the Interaction Space (IX), in order to carry on his UT with the *wisdom* derived from the enterprise *principles* and *beliefs*, following its *ideology*. For example, if the *principle* is "solidarity" rather than "profiting", the suggestion for a person's registration for studying may privilege choosing a cheap(er) degree or degrees with a supporting scholarship, as example of a *wise* decision.

Meaning to the SDP

The principle that an *actor* (a user) never carries on two UTs sequentially (i.e. without alternating with other actor's UT) and separately, is an "axiom" that aims user performance and software development efficiency by means of inducing the reduction of the articulatory distance of the UT i.e. the user's effort [Winckler et al., 2015], and by suggesting 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 that represents their completion. The design of the BP with *essential use cases* also promotes the efficiency of the SDP, since most BPs are limited in terms of the number of *tasks* and *actors*, leading to a fluent dialogue when the BP design is presented (e.g. in A4 or A3 paper sheets) and the BP logic is visible and the *tasks* are readable from sight in a meeting table. This impact reflects concern over the people (the user stakeholder) work, promotes dialogue and also commitment regarding the achievement of the project's goal, which is inevitably related to the meaning (expressed by the *goal*) of the BPs considered in the BPI under development.

3.2.3. Interaction Space (IX)

Origin

The Interaction Space (IX) is originated from the gathering of *interaction spaces* (now referenced as Interaction Components) generated from the application of the WISDOM *Software Architectural Technique* to the User Intentions of a given UT. This led to the establishment of a Model-Driven Architecture, the Hydra Framework [Costa et al., 2007], for User Interface development, which implemented the *interaction spaces* clustering using Aggregation Spaces (AS). With the formalization GOALS, the IX is structurally defined as the space of communication between two or more UTs that happen in the same (physical or electronic) space, where the same Business Rules (BR) and Data Entities (DE) are used. It reflects the relation of the customer with the enterprise, in the same way that in the DEMO method, the *initiator* and the *executer* participate in a *transaction* subject to Action Rules and using Object Classes. The IX was identified as the solution to represent physical and electronic spaces and as a basis for the User Interface development. The Lines of Visibility and Interaction which are presented in Section 4.3.1.1. Step 1 - Service Design, may also be considered as Interaction Spaces that include BPs, DEs and *actors*.

Definition

The original IX definition is derived from the WISDOM concept of *interaction space*, as a User Interface space where the "*User Interacts with Functions, Containers and Information in order to Carry On a Task*" [Nunes, 2001]. We adapt this concept to the enterprise context by means of its generalization in order to complementarily consider the support of the UT in person, as in any of the cases the same BRs and DEs apply. Hence, we redefine the IX as "*The Space that Supports a UT*" (with the same BRs and DEs). One IX supports the interaction of two or more users in person or remotely while each one carries on his own UT. If two *actors* carry on the same UT remotely, then they are necessarily performing cooperative work [Grudin, 1994].

Meta-Model and Example

The meta-model presented in Figure 16 defines that one IX can support one to many UTs. The IX uses and includes BRs, and can also directly use and include a DE if no BRs are identified to apply in that IX. In the example, the identification of IX is architecturally deduced from the relation of UTs. They are the space that supports the interaction of two or more *actors*. The IX supports the application of BRs that apply over DEs, defining the IX Logic that "builds-up" the Enterprise Structure. The relation of IX to UTs, BRs or DEs, can be expressed along with the BP design, or by means of an integrated representation of the BP design where the physical space of the IX is represented, allowing the communications of more than two *actors* per business *transaction*, as will be presented in Section 4.3.1.3. Step 3 - Enterprise Structuring.

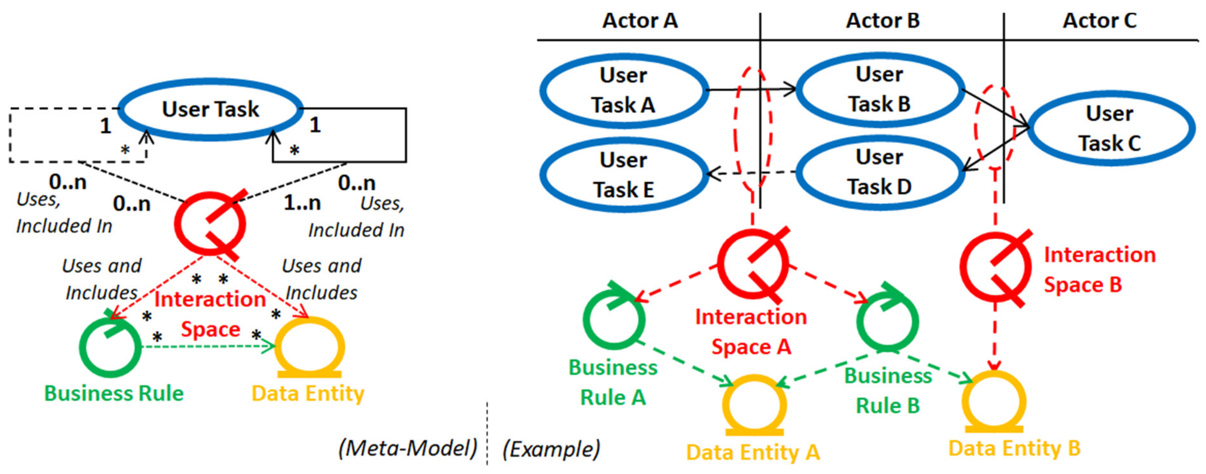


Figure 16. Meta-Model and Derivation of IXs from UTs and relation to BRs and DEs.

The example shows the derivation of IXs in order to support the interaction between Actors “A” and “B”, and “B” and “C”, by means of IX “A” (a Request) and “B” (a Coordination) respectively, which is possible since the UTs “A”, “B”, “D” and “E” are subject to the same BRs and DEs, and the same happens in the case of UTs “B”, “C” and “D”, except that UT “C” is not subject to BR “B” in order to access DE “B”. If another interaction between Actors “A” and “B” would occur (e.g. between User Task “E” and “F”), then again, the same or a new IX may be defined (e.g. “C”) in order to support that interaction, depending on the business conceptual idea.

Semiotic Structure

The IX grants the user control over information, respecting existing *paradigmatic* and the *syntactic* relations. Information should be presented using *syntagmatic* (historical) information, from which the user can withdraw *knowledge* in order to apply the enterprise *principles* and fulfill his *task* applying *wisdom*. For example, the IX can enable the possibility of proceeding with a person’s registration, presenting that person’s registrations from the past, and eventually outcomes of registrations from others to facilitate user decision.

Meaning to the SDP

The IX represents a logic that that aims user performance and software development efficiency by means of believing that the most productive way of working for the user is giving him a User Interface that facilitates his *task* in terms of *data* and tool availability. In other words, if all the needed *data* and tools are always available (or always when needed), navigation is reduced, and also the mental effort of maintaining variables that concern the *task* execution. It induces the reduction of the articulatory distance of the UT i.e. the user’s physical interaction with the User Interface [Winckler et al., 2015], and suggests the composition of the User Interface in such a way that the user has a clue of what each *concept* means, which is possible since the design is based on what are the User Intentions and User Interactions when dealing with a UT in a given system state. It also suggests locating the components by the highest affinity possible, and using as little implementation space as possible in order to raise the probability of reducing implementation objects and therefore reducing implementation effort. It can be observed as a technology-ecologic approach, that aims giving the user what he needs when he needs, using an adequate, instead of an over-worked solution.

3.2.4. Business Rule (BR)

Origin

The Business Rule (BR) is originated from the clustering of System Responsibilities (SR) derived from the application of the WISDOM *Software Architectural Technique*. Since part of the SRs had repeated semantics, and the organization was carried out by affinity of meanings, it was possible to establish a correlation with recurrent User Intentions (e.g. "Authenticate Self"), and also identify the ones which did not had a patterned and recurrent existence. These were SRs which usually characterized relevant perspectives of the BP e.g. "Calculate Degree Average", which may be only a single User Intention and Interaction, but which implementation is not patterned, is usually more complex as it expresses specific business *regulations*, whereas the remaining SRs where usually implemented in a patterned and simpler way (e.g. direct or filtered SQL queries to tables). This led to the isolation of these SRs as Business Rules highlighting the difficulties of implementation. With the formalization of GOALS, a relation was established with the DEMO Transaction' Action Rules which coordination logic however does not apply, as it is already expressed in the BP design. By means of relating the BR to the IX, we also solve an identified DEMO problem, which is when to execute the Action Rule. In GOALS, the execution of the BR is considered permanent throughout the Information System every time the containing IX is used (invoked) by other components.

Definition

The Business Rule (BR) specification is based on DEMO concept of Action Rule, which defines a structure of decision (using pseudo-code) that applies restrictions to Object Classes concerning the execution of business *transactions*. These restrictions are considered in GOALS as *paradigmatic* relations which are applied to the *syntactic* relations of Data Entities (DE) in order to produce a new valuable *business concept*. Hence, we define a BR as "*A Restriction over DE's Structural Relations*". BRs represent *regulations* or explicitly defined *requirements* that should be elicited in order to understand the restrictions which the user is subject to when carrying on a UT, and do not represent collaboration impositions with other *actors*.

Meta-Model and Example

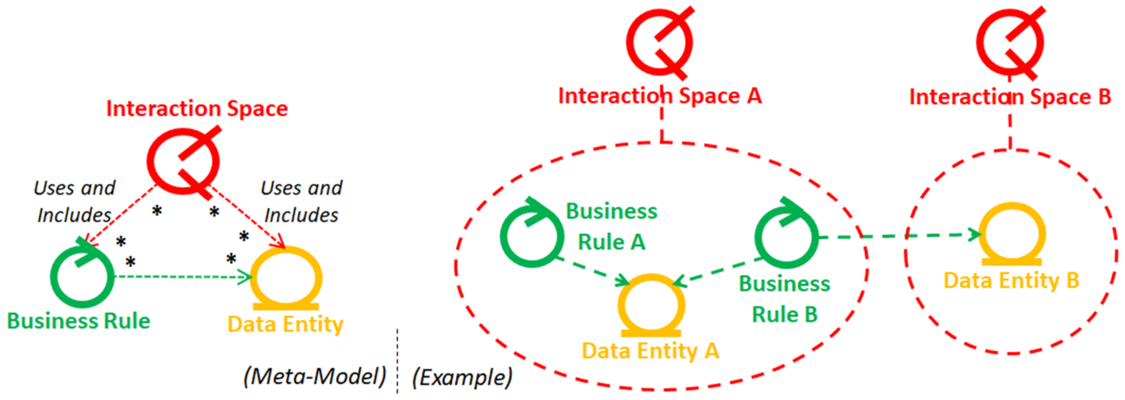


Figure 17. Meta-Model and Example of Business Rule (BR).

Figure 17 presents the BR meta-model, which defines that the BR may be used and included in many IXs, and also that a BR can use many DEs. The IX can optionally be linked to DEs. The example shows that IX "A" includes BRs "A" and "B", and that IX "B" uses DE "B", directly, without BR intervention. This is same structure presented in Figure 16, now with the physical representation of the IX.

Semiotic Structure

A BR defines a *paradigmatic* relation of *business concepts* (DEs) which cannot be structured by the DEs *syntactic* relation, resulting in a new meaning (*business concept*). This information can be presented to the IX as a *syntagmatic* structure defining an historical sequence of *data*, complementarily provide information from similar past decisions in order to support the fulfilment of UTs with enhanced *knowledge* improving the possibility of doing it with *wisdom*, and according to the enterprise *principles*. For example, if the BR should verify adequacy of a Contact for a Person, then it should apply an algorithm that will produce new information to be interpreted by the system, kept by the DE and represented in the IX (to support the UT).

Meaning to the SDP

The BRs are considered as the more important SR of the GOALS Business Logic, as they are the only business-specific programmed class in this layer. By means of isolating BRs, the *Software Architect* is able to better control the effort that will be applied in the implementation, or at least increase the possibility of acknowledging in advance where unexpected effort may be generated. This enables a more balanced management of the team, since developers may have vocation for more or less patterned work. The BR is architecturally related to the IX, however, it may be invoked by any software-specific component to ensure the conceptual specification of the Enterprise Structure. The Business Logic structuring is presented in Sections 3.3.4. User Interface System Responsibility (UISR), 3.3.5. Database System Responsibility (DBSR) and 4.3.2.2. Step 6 – Business Logic Structuring.

3.2.5. Data Entity (DE)

Origin

The Data Entity (DE) is originated from the modeling of the Information System Domain Model using UML classes, following the WISDOM method, with no changes. As it was a concept that expressed information that always had a meaning within the enterprise and was recognized by those whom have knowledge about it (stakeholders), it became a standard of communication for users, *Service Designer* and *Software Architect*, especially about the Customer, leading to the normalization of the enterprise concepts (e.g. candidacies and pre-registration become only candidacies), a “dialect” of the enterprise that may promote communication and development effectiveness. Since the Domain Model could be transformed into Database related tables, the DE domain model stands as a common development artefact.

Definition

The Data Entity (DE) definition is provided by WISDOM definition of *entity*, which is a class of “*Persistent Information about a Business Concept*”. This means that persistency will be maintained by the Information System, and that it will enclose meaningful *business concepts*. DEs are related between each other allowing a simple representation of reality which is made available by means of a Database application. In terms of common Database objects, DEs are expressed as Tables, and attributes are expressed by Fields [Ling et al., 1981]. DEs are related between each other by means of semiotic *syntactic* relations, which are expressed using a UML association, also implying the definition of multiplicity between the related DEs. Multiplicity will typically be of one-to-many or many-to-many, since the definition of a specific multiplicity (e.g. 1 to 5) is uncommon, and should be expressed by a BR due to its volatility (as it will eventually change over time). The definition of relations of one-to-one is also uncommon as in those cases the DEs meanings can usually be conciliated in a single DE.

Meta-Model and Example

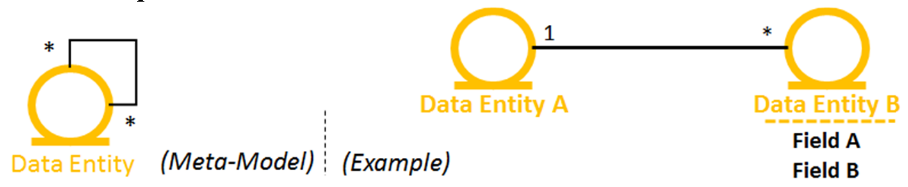


Figure 18. Meta-Model and Example of Data Entity (DE).

In Figure 18 the meta-model of the DE follows what is specified by UML and the Domain Model for the establishment of relations between classes [Booch et al., 2005]. It states that the DE can be related to another DE with an association relation with cardinality of one-to-many or many-to-many, and that a new association class can be generated from the relation of two DEs. The example presents a situation where DE “A” is related in one-to-many to “B”, which has Fields “A” and “B” (e.g. a “Student” has many “Grade” which have a “Mark” and a “Date”).

Semiotic Structure

A DE is a *business concept*, and has a meaning for itself within the enterprise which can be understood by every stakeholder (with enough expertise about the enterprise). DEs are also related between themselves, composing a simple *syntactic* structure. For example, consider: a “Person” whom has “Contacts” and is “Registered” (as a customer).

Meaning to the SDP

The identification of DEs should be carried along the BP improvement, so that the analyst may develop a well-defined notion of the concepts involved in it and how they are related. By means of establishing a dialogue based on the Domain Model, the *Software Architect* will have a very detailed vision of the *concepts* which are used along the BP execution, so that that knowledge may be transmitted to the software development team. By means of a dialogue which is based solely on existing DEs, it may be possible to delineate the construction of every component of the system, which is particularly important for the *Software Developers* of the Business Logic and the Database, and also for the *User Interface Designer*, since the User Interface is also designed using the same *concepts*. This means that the way in which the concepts are related must be “solid” i.e. almost unchangeable in time, and the only way in which this can be achieved is by means of recurrently testing the conceptual structure before the design and implementation of the system, since a misconception at the Database layer will almost inevitably lead to a restructuring of the Business Logic and the User Interface.

3.3. Software Architecture Concepts

The Software Architecture for a given Business Process (BP) inherits the Enterprise Structure based on each User Task (UT) of the BP, and is elaborated by means of the application user-centered techniques that gradually develop its User Interface, Business Logic and Database, based on the Interaction Spaces’ (IX) Business Rules (BR) and Data Entities (DE) of the Enterprise Structure. The IX, BR and DE are hybrid business and software concepts which are complemented with software-specific components to “build-up” the Software Architecture.

The Software Architecture is composed by the software-specific concepts of Aggregation Space (AS) [Costa et al., 2007] (one per UT), which is composed by Interaction Components (IC) which are composed by Interaction Objects (IO), which trigger User Interface (UISR) and Database System Responsibilities (DBSR), which architecturally use IX, BRs and DEs associated to that UT, ensuring traceability between business and software modeling.

The reuse and generalization of the identified components by means of their semantic meaning along the Analysis and especially in the Design phase, is crucial, as it promotes simplification by means of generalization, reducing analysis mistakes and eliminating duplicated artefacts. The User Interface is composed of the IX, the AS, the IC and the IO implementation concepts, and the Business Logic is composed of the BR, UISR and DBSR programmable classes' concepts. The Database is composed of Tables and Fields derived from the identified DEs. The software-specific Software Architecture concepts will be presented concerning its origin, definition, symbol, and logic of relation to other concepts, as follows.

3.3.1. Aggregation Space (AS)

Origin

The Aggregation Space (AS) is originated from the Hydra Framework [Costa et al., 2007], and represents the composition of WISDOM *interaction spaces* which support User Intentions. Since most of the User Interfaces for the support of a User Task (UT) can be implemented in a single screen without navigation (without opening or replacing the actual screen by a new one), or in the worst case, at least the initial information related to the *task* "fits" one screen, the AS conceptually became the perspective of one *actor*, when performing a UT, over the set of *data* and *regulations* of an IX, concerning one execution of a BP.

Definition

The original definition of the AS comes from its mapping to *use cases* [Costa et al., 2007]. Since most User Interfaces can be implemented in a single AS, we define the AS as "*A User Interface*", meaning what the user is watching at a given moment in the Information System. In the cases which may be necessary a second AS, the usage of one AS to another represents a navigation between both.

Meta-Model

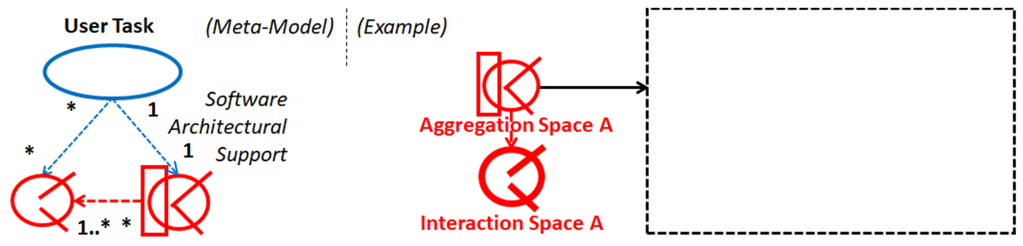


Figure 19. Meta-Model and Example of Aggregation Space (AS).

Figure 19 presents the meta-model, where it can be read that the AS provides architectural support to the UT in a relation by default of 1 UT to 1 AS. The AS uses one or more Interaction Spaces, depending on the Enterprise Structure specification. The example presents the visual representation of the AS, which is the User Interface space that the user observes, and which will be structured with Interaction Components (IC) and Interaction Objects (IO).

Semiotic Structure

The AS is a composition of ICs which should make clear to the user as much as necessary, the *syntactic* (relative to DE) and *paradigmatic* (relative to BR) *data* relations of the *business concepts* which are presented in the User Interface. The presentation should provide the user as much *syntagmatic* (relative to historical) information as possible, so that the user, being aware of the enterprise *principles*, may perform his UT as *wisely* as possible.

Meaning to the SDP

The AS is one perspective of one user of the IX when preforming a UT. This means that the more UTs that depend on one IX, the more AS it will have associated. In one group of AS associated to UTs (e.g. a UT "request" and UT "response") of the same IX, the reuse of software-specific components is very high, if not total, since what varies is the behavior of the AS, IC and IOs, and not necessarily the *data*. In each of these perspectives, which may be related to one or more Interaction Spaces (IX) depending of the UT, each *actor* (e.g. the "requester" and the "respondent") will have distinct privileges over the existing *data* and tools e.g. one *actor* is not able to edit another *actor*'s *data*. The distinct privileges, information and tools that will be available are designed following a user-centered sequence of techniques that will be presented on Sections 4.3.1.4 and 4.4.4 (Step 4 - Task Modeling).

3.3.2. Interaction Component (IC)

Origin

The Interaction Component (IC) is originated from the WISDOM *interaction space* as the User Interface space that supports a User Intention and triggers a System Responsibility. In each IC where introduced the Fields of the DE which match the user needs when performing the intermediary steps of the UT (the User Intentions), becoming the visual design tool of the User Interface using CANONICAL ABSTRACT PROTOTYPES (CAPs) [Constantine et al., 2003], since its internal organization and positioning represented a sequence of reading that facilitates the user work. The IC was also implemented by the Hydra Framework, which structure included the CAP classes which provided the needed behavior in terms of *data* management.

Definition

The original definition is provided by WISDOM as the space where the "*User Interacts with Functions, Containers and Information in order to Carry On a Task*". We define the IC as a "*Tool of a User Interface*", meaning that it holds the *data* and objects which may be necessary to execute the *actor*'s User Intention.

Meta-Model and Example

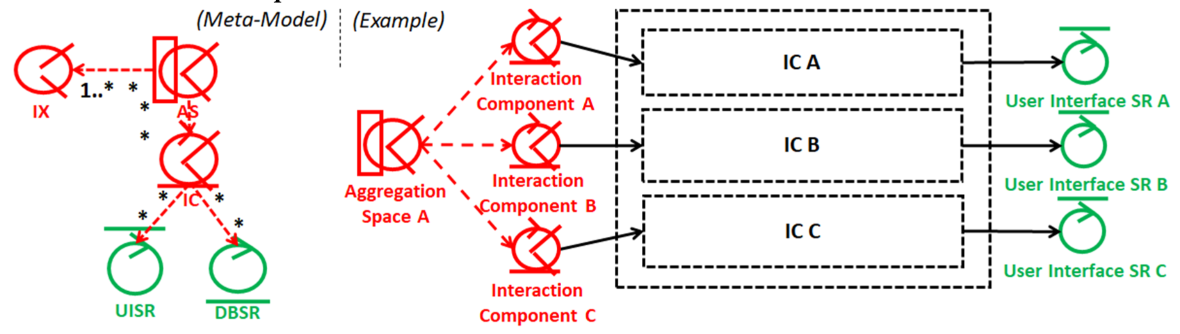


Figure 20. Meta-Model and Example of Interaction Component (IC).

Figure 20 presents the meta-model of the IC which defines that the IC is used by an AS, supports a User Intention, contains Interaction Objects (IO) and triggers User Interface (UISR) and (DBSR) Database System Responsibilities. The example presents the composition of Aggregation Space A with ICs "A", "B" and "C" (each one supports a User Intention), which trigger User Interface System Responsibilities "A", "B" and "C".

Semiotic Structure

The IC is the space of the User Interface that presents the *data* Fields organized according to the simple *syntactic* and more complex *paradigmatic* relations of the presented DEs. The presentation of *syntagmatic* relations is more complex, since it involves the historical presentation of *data* according to the *syntactic*, *paradigmatic*, or combined, presentation of *data* and past decisions along time, and may involve one of more Fields (Interaction Objects).

Meaning to the SDP

The IC is the "path" to the internal architecture of the Information System by means of the relation established to the System Responsibilities (SR), providing *data* and the expected system behavior. The supportive pair IC-SR may be further decomposed by an Interaction Model, in which case the already identified SRs serve as an interface function to new SRs. This opens the possibility of deciding the level of decomposition needed from the design in order to further structure or not the architecture beyond the IC i.e. if the tradeoff of investing more time in the design will have a revenue in terms of implementation effort or enhanced architectural control. The IC detail is presented in Sections 3.3.3. Interaction Object (IO) (the next section), 4.3.2.1 and 4.4.5 (Step 5 – User Interface Design).

3.3.3. Interaction Object (IO)

Origin

The Interaction Object (IO) is originated from the CAPs [Constantine et al., 2003] and Hydra Framework [Costa et al., 2007], representing an object of interaction which holds *data* or provides the execution of a command, triggering User Interface or Database System Responsibilities. Typically, the IO represents e.g. HTML input, select, text area, option and button elements. The IO enables User Interface and implemented code traceability.

Definition

The original definition of the IO comes from the Hydra Framework that specifies it as the Canonical Abstract Prototype that supported a "leaf" action, and interaction as identified by the CONCUR TASK TREES method [Paternò, 1999]. We define an IO as "*A User Interface Object that triggers SRs*", meaning that it is an object which is presented in the User Interface, and that establishes a direct relation to the Business Logic of the Information System.

Meta-Model and Example

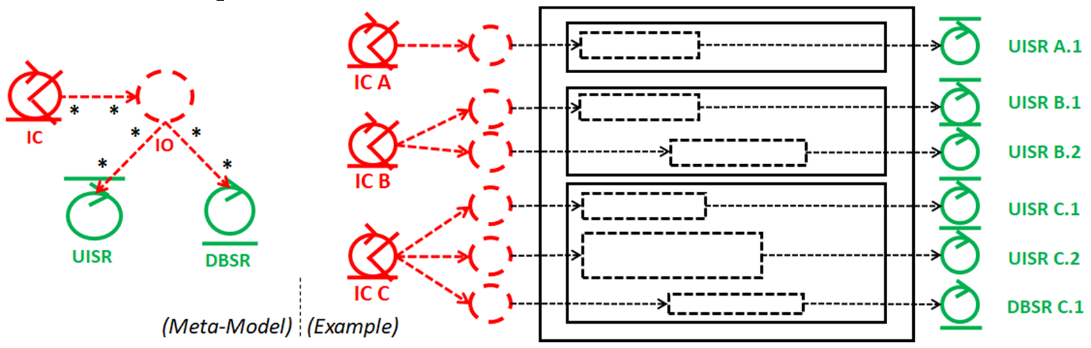


Figure 21. Meta-Model and Example of Interaction Object (IO).

Figure 21 presents the IO meta-model that specifies that each IO supports a User Interaction which is part of a User Intention which is supported by the IC to which that IO belongs to, and will trigger User Interface (UISR) or Database System Responsibilities (DBSR). The example presents the Aggregation Space composition by: IC "A" which triggers the User Interface SR

"A"; IC "B" which triggers the User Interface SRs "B.1" and "B.2"; IC "C" which triggers the User Interface SRs "C.1" and "C.1", and Database SR "C.1"; by means of the Interaction Objects presented in the User Interface. The IOs may be further specified using Canonical Abstract Prototypes (CAP) [Constantine et al., 2003], as will be presented in Sections 4.3.2.1 and 4.4.5 (Step 5 – User Interface Design).

Semiotic Structure

The IO is the space that presents the *data* Fields of *syntactic* (relations of one or more DE) and *paradigmatic* (relative to BRs) relations, including *syntagmatic* information or not. One example of a *syntactic* information is a Field of the Database which may assume a value or a list of values. A *paradigmatic* relation implies two of more DE, and when it is relative to BRs, it involves a validation. One example is a success or error message. And *syntagmatic* relations must present the evolution of the *business concepts* (DE or BR) along time. One example is a list of the evolution of the value of a DE Field over time.

Meaning to the SDP

The IO represents the detail level which is necessary in order to design the User Interface of the system. It is based on the IOs that is carried out the more artistic perspective of the Information System development, as designers are expected to organize each object in such a way that the whole (the AS which holds the ICs which hold the IOs) reflects the *data* which is being managed and that the operations and what they represent for the UT are clear to the user in order to promote its performance and minimize user mistakes.

As a guideline of design, the grouping by sequence of User Intentions (of ICs) and affinity of IOs is a base that later will enable introducing the artistic facet of *User Interface Designers*, who will express and develop its competences by means of the User Interface design with the added value of knowing in advance, in a structured way, what are the business *regulations* and the *data* concepts which are being elaborated, and how the user expects to achieve his *task* objectives. The User Interface design is further presented in Sections 4.3.1.4. Step 4 - Task Modeling and 4.4.5. Step 5 – User Interface Design.

3.3.4. User Interface System Responsibility (UISR)

The User Interface System Responsibility (UISR) is originated from the need to represent a programmed function that provides *data* for Interaction Components (IC) and Interaction Objects (IO). It represents a specific type of Business Logic function implementation which characteristic is the provision of *data* to fill one or more IOs. The UISR may also execute operations over the Database, in the same way that a Database System Responsibility (DBSR), so that it is able to register e.g. patterned information of user, *data* and IP address. In fact, both the UISR and DBSR have the same structure, but due to the nature of the work which is carried out in terms of software development, the distinction is important, as it may have a direct impact in the software development team organization, reason why the conceptual distinction of UISR and BDSR stands as a form of promoting software organization.

Definition

The original definition of the System Responsibility is provided by ACTIVITY MODELING, as the system response to the *essential use cases* User Intention. That definition is adopted and extended in order to specialize the User Interface SR for the provision of *data* for an Interaction Object or an Interaction Component, as "*A SR that provides support for User Interface data presentation*".

Meta-Model and Example

Figure 22 presents the UISR meta-model that states that the UISR is triggered by ICs or IOs, and that the UISR invokes DBSRs, other UISRs, and DEs. The existence of a relation between UISRs reflects a situation where the invoking IC' UISR serves as a (function) interface to the existing IO' (UI or DB) SRs. The inexistence of this relation means that the UISR has to assume the responsibility of collecting the *data* and executing the necessary Database commands, since this responsibility is not further divided (modularized) in IOs and corresponding (UI and DB) SRs. The example presents the structuring of the Business Logic composition with UISRs, namely the ones which were derived from User Intentions and use the UISRs which were derived from their User Interactions: the UISR "A" which uses UISR "A.1"; the UISR "B" which uses UISRs "B.1" and "B.2"; and UISR "C" which uses UISRs "C.1" and "C.2".

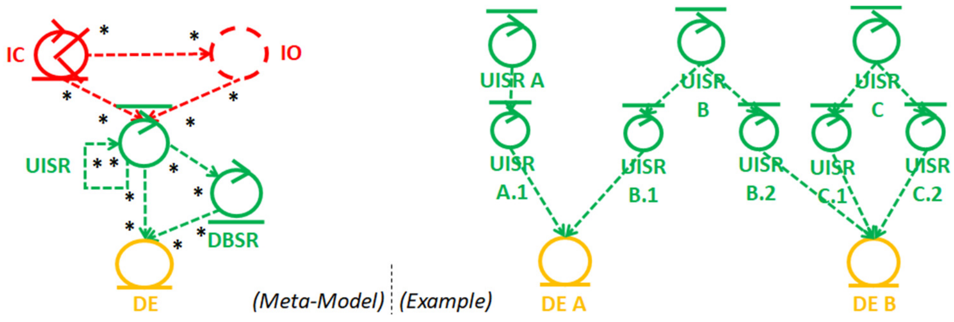


Figure 22. Meta-Model and Example of User Interface System Responsibility (UISR).

Semiotic Structure

The UISR is responsible for the transfer of *syntactic*, *paradigmatic* and if possible *syntagmatic data* from the Database to the User Interface.

Meaning to the SDP

The UISR represents the simplest form of implementation of Business Logic functions, since the collection of *data* can be regarded as the simplest way of managing *data*, also known as a "reading" act. It is also the SR which usually exists in bigger number, as User Interfaces usually have more *data* Fields (e.g. input, select, text area, option) than commands (buttons). For this reason, concerning the development of an AS, it may be more productive if the same developer is responsible for the implementation of every UISR, as he will have to manage simple yet a large number of functions. By means of maintaining control of the implemented Business Logic functions, the Project Manager is also able to control the evolution of the development, knowing in advance that the UISRs will typically take less time to develop than DBSRs and BRs.

3.3.5. Database System Responsibility (DBSR)

Origin

The Database System Responsibility (DBSR) is originated from the need to represent a system function that alters *data* in the Database, typically to record the results of the UT. It represents a type of Business Logic function which characteristic is managing large amounts of variables that have to comply with business *regulations* by means of implementing and validating them, whether these *regulations* (Business Rules) are implemented as a separated Business Logic function or not. The BDSR may also provide *data* to the User Interface in the same way that a User Interface System Responsibility (UISR) does, in order to e.g. provide immediate feedback. Both UISR and DBSR have the same logic structure, but due to the nature of the work which is carried out, the distinction is important, as it has a direct impact in the team organization.

Definition

The original definition of the System Responsibility is provided by ACTIVITY MODELING, as the system response to the *essential use cases* User Intention. That definition is adopted and extended in order to specialize the Database SR for the management of *data* as triggered by an Interaction Object or an Interaction Component, usually in order to implement algorithms that prepare the *data* which will be recorded in the Database, respecting the existing BRs, since they are always architecturally subject to the validation which they provide. Hence, we define the DBSR as "*A SR that manages data*".

Meta-Model and Example

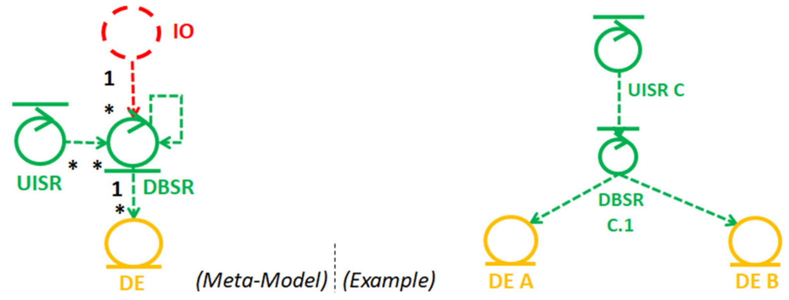


Figure 23. Meta-Model and Example of Database System Responsibility (DBSR).

Figure 23 presents the DBSR meta-model that states that the DBSR is triggered by ICs or IOs, that the DBSR invokes UISRs, and other DBSRs and DEs. The existence of a relation between DBSRs reflects a situation where the invoking IC' DBSR serves as a (function) interface to the existing IO' (UI or DB) SRs. The inexistence of this relation, in a same way that happens with the UISR as reference in the previous Section, means that the UISR has to assume the responsibility of receiving the *data* and executing the necessary Database commands. The example presents the DBSR "C.1" which supports a User Interaction of the UISR "C" (which on its turn supports a User Intention).

Semiotic Structure

The BDSR is oriented to the application of *paradigmatic* algorithms to the received *syntactic* information in the process of transferring *data* from the User Interface to the Database.

Meaning to the SDP

The DBSR potentially represents the more complex form of implementation of Business Logic functions, as usually these functions receive the information that corresponds to every object in the User Interface and the actual state of the system, and processes it respecting existing BRs. The DBSR typically exists in a smaller number than UISRs, but its implementation may represent more development time than the implementation of all the UISRs that support a UT. For this reason, concerning the development of a given AS, it may be important that the person that develops DBSRs is not responsible for the remaining UISRs that compose the Business Logic, since the overhead which is generated may be too much for the organization capabilities of a single person, and therefore reduce the overall software development performance.

By means of maintaining control of the implemented Business Logic functions, the Project Manager is also able to control the evolution of the development, knowing in advance that the DBSRs will typically take more time to develop and change than UISRs, since if not implemented in rigor the situation (an eventual "bug") will always be noticed, if not during testing, in production, which is the worst case possible.

3.4. Conclusions

This chapter presented the notation of the language by means of the presentation of each concept definition, symbol and meaning, providing insight concerning the relation of each with the remaining concepts of the language, so that the modeler may analyze each one by itself. It allows understanding the logic which is associated to the concept, what it expresses, and what does it means when it is related to other concepts, including the number of other concepts on which it relies on (uses) and sustains (is used by), so that the modeler may understand the architectural logic of its enterprise, the business, the software, and the relation between both.

By explaining the origin we explain how each concept was evolved along time and its meaning to the SDP, we are also providing insight on the importance of each concept at each moment, and how the process may follow, since the existence of each concept suggests the continuation of the SDP in the "direction" of the Software Architecture finalization. By providing the semiotic structure of each, we are complementing the meta-model and example, in order to provide business and software constructive guidelines: *ideology*, *beliefs*, *principles* and *wisdom*, more closely related to the business model; and, *knowledge*, *syntagmatic*, *paradigmatic*, *syntactic* and *business concepts*, more related to the software model.

The GOALS process and method that apply techniques to use the language concepts is presented in the next Chapter (4. The Goals Process, Method and Techniques).

4. The Goals Process, Method and Techniques

GOALS has a conceptual structure and a notation that provides a language with which business and software are modeled. This language is used to model the Enterprise Structure and the Software Architecture following the GOALS process which is composed of the Analysis and Design phases. The Analysis and Design phases apply a single method of 8 steps, 4 of which are relative to the Analysis phase and the Enterprise Structure elaboration, and 4 of which are relative to the Design phase and the Software Architecture elaboration. The strategy of enterprise development is based on the evolution of the enterprise Interaction Spaces (IX). The GOALS process, strategy, method and techniques are presented as follows, including the presentation of two case studies ("iKiosk" and "Star Project"): the first oriented to the application of the method, each step origin, elaboration, meta-model, and meaning to the SDP; and the second oriented to the aspects that should be focused in each step in order to achieve its correct elaboration.

4.1. Process

The process has two phases: the Analysis and the Design, as two sequential (sub) processes. In the perspective of the *Business Owner* they represent the conception of the enterprise, its business and the relation with the customer. In the perspective of the *Enterprise Architect*, they refer to the analysis of the business problem, and the design of the software solution for a given Business Process (BP) Improvement (BPI) concerning the strategical needs of the enterprise. The phases may be elaborated independently, as once the Analysis is finished, the Design may be carried out, immediately or posteriorly, as long as business *regulations* and *business concepts* are maintained.

The modeling of both phases is carried out using the language for both the business and software models. The language concepts refer to business (business-specific), to the software (software-specific), or both (hybrid). The hybrid concepts are concepts of the business model which are part of the software model, which are the Interaction Space (IX), Business Rules (BR) and Data Entities (DE), and which are complemented with software-specific concepts concerning the Software Architecture elaboration. Figure 24 presents the structure and related method phases, using the concepts already introduced in Section 3.1. Structure.

These concepts presented are:

- **Enterprise Structure**: Business Process (BP) (and *goal*), User Task (UT), Interaction Space (IX), Business Rule (BR), and Data Entity (DE).
- **Human Activity**: User Task, User Intention and User Interaction.
- **Software Architecture**: Aggregation Space (AS), Interaction Component (IC), Interaction Object (IO), User Interface System Responsibility (UISR), Database System Responsibility (DBSR).

The structure is what drives the process in the sense that all the modeling work targets the elaboration of the two structures. The concepts, and its diagram meta-model will be presented in a top-down logic following the dependency of concepts of the Enterprise Structure and the Software Architecture. The method that applies techniques that use the language is presented in Section 4.2. Method.

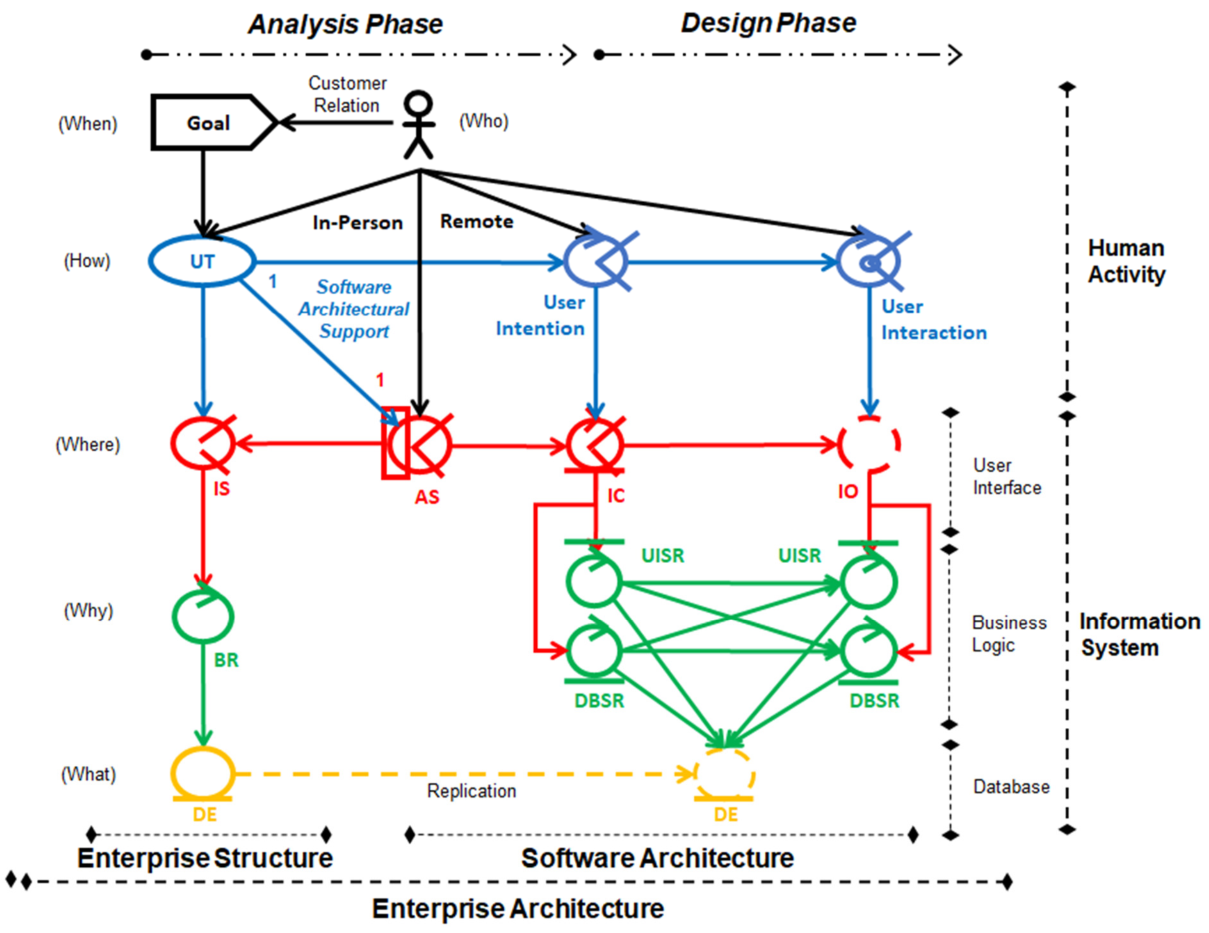


Figure 24. GOALS Process Analysis and Design Phases Modeling Concepts.

Figure 24 presents the GOALS Process and the relation of concepts which are used to model the Enterprise Structure and the Software Architecture in the Analysis and Design phases, respectively. The relation between concepts is of *dependency* (many-to-many), except for the relation of the *Software Architectural Support* (SAS). The SAS is the logic that relates UTs and AS in a relation of one-to-one, as an approach that targets implementation of the *requirements* of one complete *business process task* (UT) in a single User Interface (AS). One or more UTs happen in an IX of the enterprise, which defines the channel of communication between *actors* and their *tasks* (UTs). As these IXs implement Business Rules (BR) and Data Entities (DE), the IX is also used by the AS (the User Interface) as an architectural imposition, reflecting that the BRs and DEs of the enterprise must be respected in the communication of *actors* according to the Business Architecture.

4.1.1. Iterative and Incremental

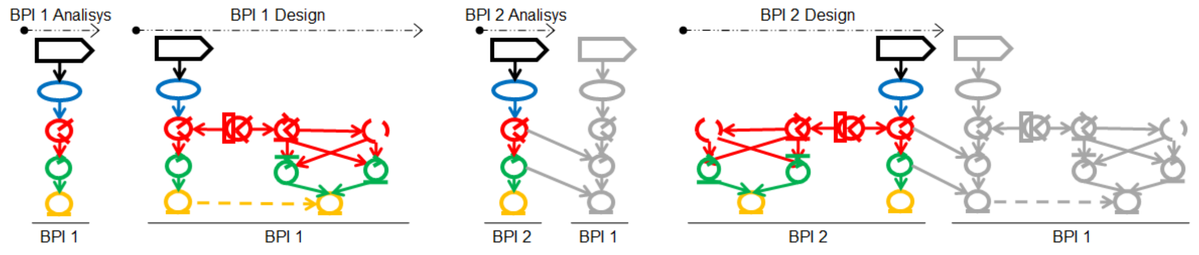


Figure 25. Goals Enterprise Development Process (EDP) Iterative Process.

The process is iterative, and further defines that upon finishing the Analysis of a BPI, it (the Analysis phase) may follow to another BP, and the design and implementation of the current BPI may take place in parallel. Once the Analysis of the new BP is finished, its Design and implementation may follow again, and again the Analysis phase follow to a third BP, gradually incrementing the modeling and implementation of the Enterprise system.

Figure 25 presents an example, in which (from left to right), first takes place the "BPI 1" Analysis elaborating the Enterprise Structure, which follows the "BPI 1" Design, elaborating the Software Architecture of "BPI 1". The process goes on to the "BPI 2" Analysis, which includes and may be related to the "BPI 1" Enterprise Structure, and follows to the "BPI 2" Design phase, which includes the models of both "BPI 1" and "BPI 2".

4.1.2. Strategy

We define as a strategy, the gradual automation of the Interaction Spaces (IX) of the enterprise, by means of transforming one physical IX (e.g. an office) into User Interfaces (Aggregation Spaces) that support the work of those who carry on their UTs in that IX. This should be carried out in function of what may be the IX which automation is more valuable for the enterprise concerning its functioning, including user (and customer) satisfaction (relative to UX), while saving *resources*. The improvement of IXs based on the automation of *tasks* which are solely made in person mat benefit the relation with the customer, and at the same time replace collaborators working time, saving human *resources* that may be used to other BPIs. As an added suggestion of the strategy, the BPs which represent bigger loads of work in specific periodic times may be automated first, in order to induce production stability.

One IX can hold the *tasks* of one-to-many (usually two) *actors*, and by means of elaborating the Enterprise Structure relative to that IX, a conceptual structure is created that allows understanding the distinct characteristics of the *tasks* which are based on it. Each *task*, which may use one to many IX (usually two). can then be modeled and the corresponding software structure designed, with the awareness of the applicable BRs and DEs, and solely the way in which *data* is managed and introduced in the system varies from UT to UT.

4.2. Method

We use the GOALS structure (Figure 26) to illustrate how each component is produced in each method steps. Steps 1 to 4 refer to the Analysis phase and Enterprise Structure, and Steps 5 to 8 refer to the Design phase and the Software Architecture. In the GOALS structure it is possible to identify that the Service Design (Step 1) and Business Process Design (Step 2) involve the BP and DE, and, BP and UT, concepts, respectively. This happens in the course of composing the Enterprise Structure (Step 3), complementing it with the IX and BR concepts using the *Enterprise Architecting Technique*. Each identified UT is detailed in User Intentions using a Task Model (Step4) and applying the *Software Architecting Technique*, as the first level of the application of the CONCUR TASK TREES (CTT) method, structuring the system in terms of available Interaction Components (user tools), which at this stage may be physical or electronic, and the corresponding higher-level UISRs and DBSRs. The Design further specifies each User Intention using an Interaction Model (Step 5), in a second level of the CTT and the *Software Architecting Technique*, integrated with the BEHAVIOR DRIVEN DEVELOPMENT (BDD) method, conceiving the User Interface, and further structuring the system with IOs, UISRs and DBSRs, proceeding towards the finish by structuring the Business Logic (Step 6), the Database (Step 7), and the Software Architecture (Step 8).

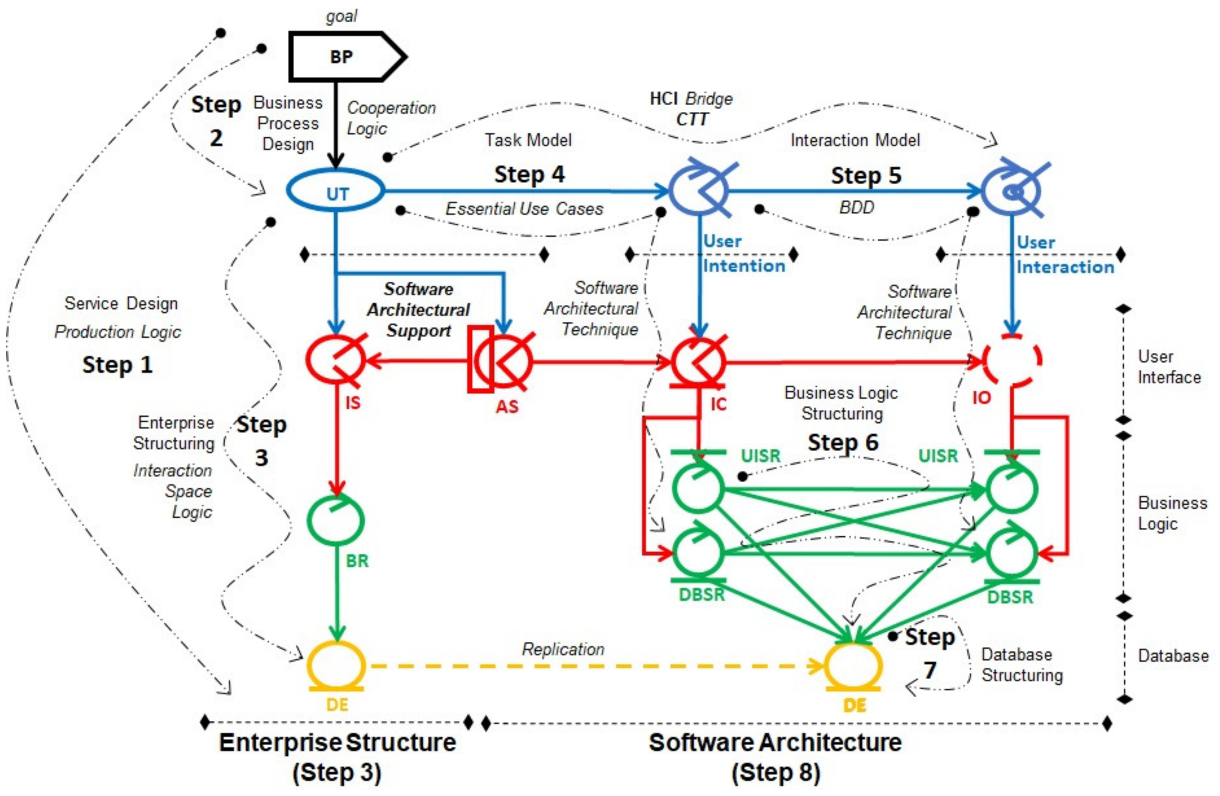


Figure 26. GOALS Method Steps (1 to 8) and suggested Techniques.

Together with the Method, the GOALS structure defines the GOALS Enterprise Architecture and the set of structured techniques that allow understanding how the whole business and software structures are elaborated (modeled). The Service Design, the Business Process Design and the Enterprise Structuring (Steps, 1, 2 and 3), apply the *Production Logic*, specifying the production of the BPs in terms of DEs, the *Cooperation Logic*, specifying the cooperation between *actors* in IXs, and the *Interaction Space Logic*, that specifies the BRs and DEs which are used in each IX. By means of the detail of the UT and the specification of a User Interface (an AS) by means of the *Software Architectural Support*, and the application of a Task Model (Step 5) and an Interaction Model (Step 6), for a *task* decomposition according to the *essential use cases* and BEHAVIOR DRIVEN DEVELOPMENT (BDD) techniques that identify all the Business Logic (BRs, UISRs and DBSRs) of the system is identified. Then, all the identified artefacts (GOALS concepts - UML classes) are structured still in a top-down process, based on the common meaning identified along the method and expressed as DEs in a Domain Model.

The GOALS method steps are processed according to the ongoing SDP that drives the project team coordination in the modeling of each diagram, using objects (instantiations) of the classes of the GOALS structure, which artefacts elaboration require valences in the domains of Enterprise Engineering, Human-Computer Interaction and Software Engineering. For this reason, GOALS maintains the compatibility of each concept so that it may be recognized by each domain practitioners and cooperate in each Step, in a progressive transfer of responsibility from the EE to the SE domain by means of HCI, focusing enterprise development, design and software implementation productivity. The cooperation starts in the Analysis and continues to the Design where users, HCI and SE specialists, conceive the final Software Architecture. EE is related to the roles of *Business Owner*, *Service Designer* and *Business Process Manager*, HCI to *User Interface Designer*, and SE to *Software Architect* and *Software Developer*.

Table 6 presents the artefacts used and produced by the method in the process of developing the business and software models. The "*Business Inputs*" provide unstructured information that must be available in any format. The "*Enterprise Structure*", "*User Behavior Specification*" and "*Software Architecture*" are artefacts that take part as inputs (I), outputs (O) or both (I/O) in the "*Output Models*" of each Step, and may also be used in the subsequent Steps.

Table 6. GOALS Method Steps, Input and Outputs, and Affinity of Responsibility.

Process	Analysis				Design				Responsibility		
Artifacts/Steps	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**EE**	**HCI**	**SE**
Business Inputs											
Business Organization	I	I							*		
Business Regulations			I						*		
Business Concepts	I	I	I	I	I		I		*		
User Collaboration	I	I		I	I				*	*	
Enterprise Structure											
Business Process (BP)	O	I	I					I	*		
User Task (UT)		O	I	I				I	*	*	
Interaction Space (IX)		O	I		I			I	*	*	
Business Rule (BR)			IO			IO		I	*		*
Data Entity (DE)	O		IO				IO	I	*		*
Human-Computer Interaction											
User Intentions				O	I				*	*	
User Interactions					IO					*	*
Software Architecture-Specific Components											
Aggregation Space		O		I		I		I		*	*
Interaction Component				O	I	I		I		*	*
Interaction Object					O	I		I		*	*
User Interface SR				O	O	IO	I	I			*
Database SR				O	O	IO	I	I			*
Output Models											
Service Model	O	I							*		
Business Process Model		O	I	I					*		
Business IX Organization			O						*	*	
Enterprise Structure			O			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			*

4.2.1. Architectural Control and Agility

The method is oriented by the strategy of defining an iterative BP-sized Analysis and User Task-sized incremental design process to which is given flexibility in the form of a trade-off between agility and business architecture, while maintaining traceability, targeting enterprise architectural control. Agility is applied to the business architecture (the Enterprise Structure), finding the room to apply user-centered techniques in the Interaction Space (IX). The trade-off is explained by means of the brief analysis of two complementary cases of improvement:

- The ones which involve the Enterprise Structure, including Service, BP and UT rearrangement, with a bigger impact in the Software Architecture, more specifically in the software-specific components derived in the Task and Interaction Models;

- The ones which are circumscribed to the Software Architecture and a single IX or a single UT (without BP reorganization) have a bigger probability of involving a smaller number of components, having a smaller impact in the Software Architecture, as all are subject to the same Business Rules (BR) and Data Entities (DE).

Thus, when the enterprise change implies BP reorganization, the method must follow the Analysis and Design phases (Steps 1 to 8), otherwise it may be started from the Design (Steps 5 to 8) upon User Intentions validation (Step 4). This defines the agile trade-off characteristic of GOALS, in which the time invested in the first 3 (Steps) business models (usually expressed in one A4/A3 diagram per model depending on the BPI), has the revenue of the availability of a business requirements structure (the Enterprise Structure/Business Architecture), that becomes the "back-bone" of the Software Architecture by means of the application of user-centered techniques, in a "top-down" process, using the already related business-and-software hybrid concepts of IX, BR and DE, to apply architectural interaction patterns that fully structure the User Interface, Business Logic and Database of the Information System.

4.2.2. Architectural Interaction Patterns

GOALS derives an architectural Human-Computer Interaction (HCI) pattern in function of the interaction design in two steps (4 and 5), which conceive the two layers of User Interface and Business Logic, for the support of the human User Intentions and Interactions, respectively. The pattern defines that every User Intention will be supported by an Interaction Component (IC) and a single UISR which will be responsible for receiving and sending *data* to and from the Database, and which will execute other UISRs and DBSRs as a support for User Interaction. The second-level UISRs and DBSRs are derived according to the need to build information structures that the system may present to the user, so he may see and understand, followed by the execution of *data* validation algorithms, according to the existing BRs and the DE' structure by means of the invocation of the IXs on the AS is based. Alternatively, BRs may be invoked by any other software-specific component to promote architectural and/or SDP optimization. In concrete terms, this User Interface may be a web page or client-side application that loads and presents all the *data* in fields, which behavior is implemented by the AS, and which saves the *data* to the Database upon user command. The pattern is called as "Simple – Interaction Design" because the system immediate feedback is not enforced, and because it defines specific architectural support for each User Interaction. The pattern is presented in Figure 27. Other possible interaction patterns are presented in Appendix B – Interaction Patterns.

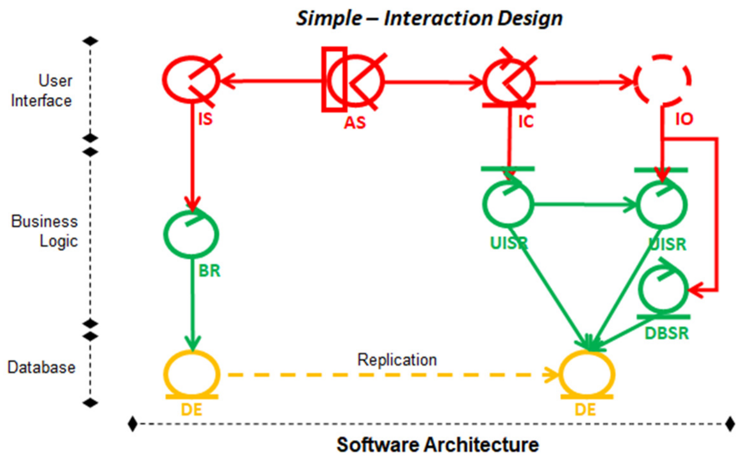


Figure 27. "Simple - Interaction Design" Pattern.

4.3. Case Study – “iKiosk” Project

In order to illustrate the method of our approach we present a real software project developed for the conception of a new enterprise. This business has the objective of selling classifieds (announcements) and presenting actuality news in Madeira Island, Portugal. The project is named as “iKiosk” since this type of information is usually available in newsstands (kiosks) where coffee is sold daily. The idea is to settle a local store where the information will be available in electronic format and only optionally in printed paper, serve coffee and provide in-person support for classified and news printing and introduction. The system will also be available for the World Wide Web. The *Business Owner* settled important preliminary societies with “local” reporters who wished to post the news of their specialty. The project has increased importance, and takes advantage, of the fact that printed journalism in Madeira is quite restrict and also suffered a drawback in recent years, as one of the two more important journals found its sponsoring cut due to the austerity measures applied in Portugal after 2010. For this reason, the *ideology* of the project is: “*Belief* in the revitalization of the news service in Madeira by means of on-line publishing based on the *principle* of the freedom of speech of qualified reporters”. Three phases were defined, which were named as follows:

- **Business Modeling**. The modeling of the parts of the business parts that would support the classifieds and news management, so that the *Business Owner* could understand the complexity of the project which he had idealized.
- **Software Modeling**. The modeling of the Information System targeting the elaboration of the website and the User Interface of the *tasks* to support its information, so that a *Software Developer* could acknowledge all the logic behind the conception of the system.
- **Implementation**. Two possibilities were defined concerning implementation. Depending on available funds, it would take place in parallel with the installation of the store facilities. Otherwise, only the website would be implemented.

The business modeling was carried out, and the roles involved were: the *Business Owner* who accumulated the roles of *Service Designer* and *Business Process Manager*, the *Software Architect* and a *User Interface Designer*. The main produced diagrams, presented in this Chapter are:

- Service Design (of the enterprise).
- Business Processes Design of:
  - “News Decoy”
  - “Publish Advertisement”
  - “Publish Classified”
  - “Validate Reporter”
- User Tasks
  - “Insert Classified” and “Obtain Classified” of the “Publish Classified” BP
  - “Obtain News” and “Publish Reporter News” of the “News Decoy” BP.

These previous diagrams are presented as follows in order to illustrate the GOALS method steps and the applied techniques. The remaining diagrams are presented in Appendix C – “iKiosk” Project Diagrams.

4.3.1. Analysis Phase

The Analysis phase models the enterprise in terms of a working system, which is independent from the implementation-technology of the supporting Information System. The enterprise system defines a human and logical structure that provides a service that has *regulations* and manages *data* based on an Information System (independently of its physical implementation).

This working structure is represented by the Enterprise Structure (elaborated in Step 3), which is based on the Service (Step 1) and Business Process (Step 2) designs, to which a fourth Step is added in order to elicit the daily tools (based on User Intentions) that the user needs in order to carry on his *tasks*, so that business and software modelers are aware of the logical context in which user carries on its *tasks*. Step 4 (Task Modeling) establishes a first direct relation between business and software models, to which the software Design phase follows. The four Steps of the Analysis phase are presented as follows, starting with the Service and Business Process design (Steps 1 and 2), the Enterprise Structuring (Step 3) and the Task Model (Step 4).

4.3.1.1. Step 1 - Service Design

Origin

The Service Design is originated from the need to understand the relation between Business Processes (BP), user participation, and used and produced *resources* and information, reflecting the *Production Logic* of the enterprise. It enabled understanding the specificities of the desired BP Improvement (BPI), representing the inception on the business problem after the specification of the *ideology* of the project based on the High-Level Concept [Kreitzberg, 1996], as also used in the WISDOM method. The established relation is useful for the *Business Owner*, *Service Designer*, and *Software Architect* in order to rationalize over the BPI using a single diagram. Moreover, the relation between DEs which is established in this Step is also included in the software-oriented domain model of Step 7 (Section 4.3.2.3. Step 7 – Database Structuring), as previously introduced in Section 3.2.5. Data Entity (DE).

Elaboration

The *Production Logic* follows the BP meta-model, stating that: A BP must be initiated and other users may also participate, and that at least a DE must be produced, directly reflecting the semantics of the *goal* of the BP or not, to which, however, it is always architecturally associated. BPs do not relate directly between each other. This relation is established by people and DEs. We assume that the number of produced *resources* (produced DEs), should be increased against the number of consumed *resources* (used DEs) in order to enhance productivity. The used and produced DEs are represented in the diagram.

The Service Design is a set of interrelated BPs by means of DEs and *actors* that provide a service according to the business organization, including the lines of interaction and visibility. We define the Service as "*A network of BPs that provides a Service to the Customer according to the Enterprise Goals*". It is the expression of a service which is available in different touch-points, with back-end support, which the *actor*, may visit, but not necessarily in a pre-defined order. For this reason, the participation of the initiator of a BP is considered as Conditional i.e. it is initiated only if the user wants to initiate it, whereas the participation in the BP may be considered as mandatory or Sequential, since it has to give “sequence” to the initiation, otherwise, the BP will not be properly executed. The concepts of Conditional and Sequential participation will also applied in the Business Process Design (Step 2) in order to specify the (mandatory or not) flow between BP tasks.

The elaboration is carried out by means of the identification of the BPs with which the *actor* interacts with, and the identification of the used and produced products and services which are represented by DEs. BPs are named according their *goal*, their initialization (represented by a line with an arrow) and participation (line with no arrow) by *actors* is specified, and the used and produced DEs are identified (produced DEs should be explicitly identified). The touch-points are represented by the initialization of BPs by the *actors* placed outside the line of visibility, who may go beyond it in order to interact with the enterprise/service/*business process*, and when they do, they interact with the *actors* which are outside the line of visibility, representing a logical and physical location of the execution of the BPs that separates internal from external collaborators with the purpose of representing the Service as a whole.

Meta-Model and Example

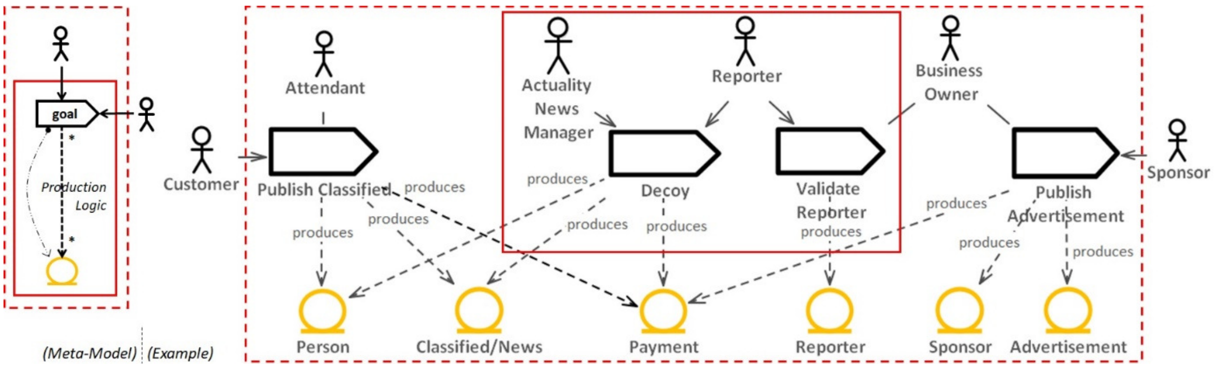


Figure 28. Step 1 - Service Design Meta-Model and Example ("iKiosk" Project).

Figure 28 presents the simplified meta-model of the Service Design, which is composed by the BP, *actor*, DEs, lines of interaction (dashed square) and visibility (internal square), to which the *Production Logic* should be applied. The example presents the relation of the "Customer" *actor* with the "Publish Classified" BP, to which the "Attendant" is also related in a relation of participation, whereas, the "Publish News" and "Decoy" BPs are initiated by the "Reporter". Two BPs use the "Classified/News" DE which was generalized in order to structure the information of both classified and news that the "iKiosk" wishes to make available to their clients, with the distinction that the news are introduced by the reporters, and the classifieds are introduced by the customers. The "Payment" DE is relative to the payments of printing of classifieds and news, and the "Person" DE represents the customer personal *data*.

Meaning to the SDP

The Service Design is the first approach to understanding how the enterprise interacts with the customer, how it is organized and what are the *business concepts* which are involved in each BP, and how they are related between each other. By establishing this relation, the analyst develops a perception of how the information may be used, and what other information may be added in order to process and obtain the desired outputs, including the BP *goal*.

The *Software Architect* will continue to explore how each BP internally works, and how it may be improved and automated in order to promote the enterprise performance. It is important that the analyst establishes the relation between DEs along the project, yet it is not mandatory at this stage, since this relation will be established along the Method Steps as introduced in Section 3.2.5. Data Entity (DE) and will be further specified in Section 4.3.2.3. Step 7 – Database Structuring.

4.3.1.2. Step 2 - Business Process Design

Origin

The Business Process (BP) design is originated from the need to understand how information is processed along a BP, what where the intervenients, and the *tasks* that they performed in order to achieve the BP *goal*, as formalized by the Process Use Cases method [Valente & Sampaio, 2007]. The simplification introduced by the normalization of User Tasks (UT) as *essential use cases* [Constantine & Lockwood, 2001], made the resulting diagram more easily readable. Thus, once designed, the model was used as a dialogue artefact in which the stakeholders expressed their concerns, namely the users when describing their *tasks*, and the *Business Process Managers* when describing the control needs of the system and the *regulations* that should be applied. The model is also useful for *Software Developers* to understand the course of user action, and *Software Architect* to define implementation priority.

Elaboration

The BP is designed by means of the relations between UTs and the human structure that cooperates until they achieve the enterprise *goal*. The relation between UTs can be Conditional or Sequential, and at least one *actor* must be defined as the initiator of the BP. The Conditional execution (dashed line), is defined as an *actor* intention to follow a certain BP path or not. The flexibility complements the traditional Sequential (mandatory) execution flow (solid line). Two UTs may be carried out by the same *actor* in the same IX only if: (i) related Conditionally, complete and meaningful, and not dependent on each other; or (ii) related Sequentially, with at least another *actor* UT between them. This situation is clear when applied to the customer in the Service Design, but it applies to BP design in a different way, as the user may not be able to accomplish certain *tasks* if their *data* does not conform existing Business Rules (BR), so it may happen that the user whishes not follow a certain path, but he is obliged to in order to add information that satisfies existing BRs. In the BP design, the *regulations*, which in GOALS are expressed as Business Rules (BR), are not represented, as they are later expressed in the Enterprise Structure. The BP design is elaborated using an Activity Diagram with UTs and swim lanes, or by relating each *actor* to his UTs, and representing the IX physical space.

Meta-Model and Example

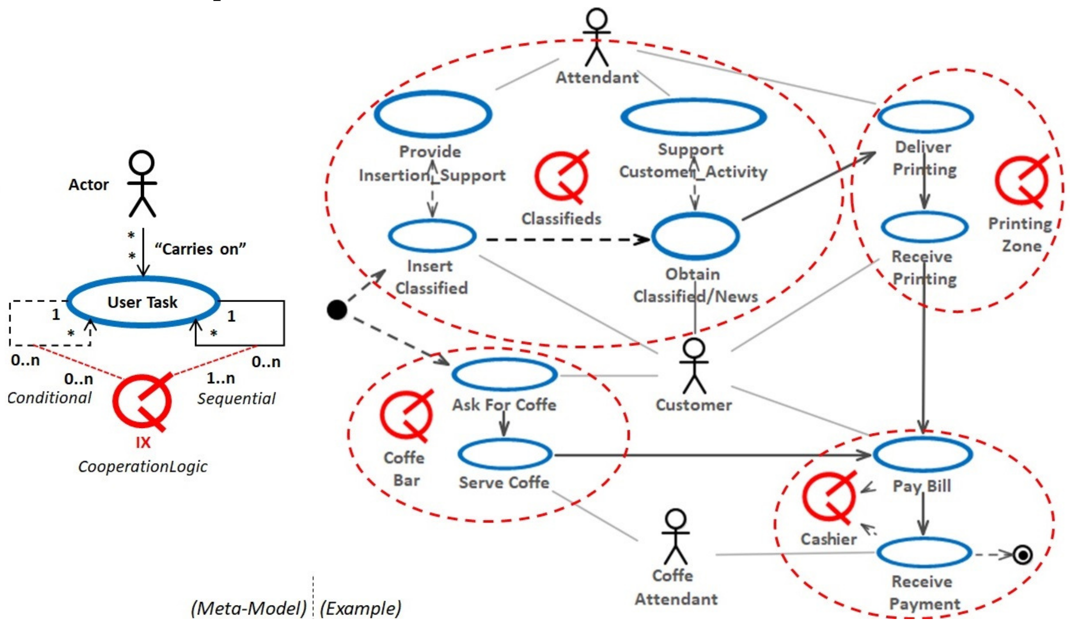


Figure 29. Step 2 – BP Design Meta-Model | Example of "Publish Classified" BP.

Figure 29 presents the meta-model of the BP design, which is composed of the *actor*, the UT and the IX. The example presents the "Publish Classified" BP design, which is composed of the UTs "Insert Classified" and "Obtain Classified/News" which are carried out by the "Customer" Actor, and receive the conditional support of the "Attendant" by means of the "Provide Classified Support" and "Support Customer Activity" UTs, respectively, depending on the will of the customer. Both customer UTs are conditional, meaning that the customer may interact with the business without completing any of the UTs. Notice that the representation of the IX physical space is not supported by UML, and for that reason, the diagram must be complemented with drawing tools (or drawn out by hand) for this representation purpose.

The "Obtain Classified/News" UT initially was "Obtain Classified" (only), but given the generalization which was carried out in order to support "Classifieds" and "News" after validation, in Steps 4 (Task Model, Section 4.3.1.4) and 5 (User Interface Design, Section 4.3.2.1), that the structure could be generalized, the "Obtain Classified" and "Obtain News" became the same. The specification of separate UTs for "Insert Classified" and "Obtain News/Classifieds" in the same IX, is a decision which targets the division of the type of support from the "Attendant" and the computers of the store where these UTs could be performed.

Meaning to the SDP
The BP design is the cornerstone of the dialog between stakeholders, including the final user. It is based on it that the *Service Designer*, the *Business Process Manager*, and the *Software Architect*, make the dialogue that will detail the system *requirements* and explicit existing expectations, to which the software modelers must comply with. The BP design will serve the purpose of defining the priority of implementation, since the UTs information may be used along the implementation namely for the integration and testing of the programmed software components. As optional requirement, we suggest that at this stage the Interaction Spaces (IX) where *actors* interact are identified, designing the physical perspective of the BP, in order to consolidate the rationale behind the elaboration of the Enterprise Structure in Step 3. Otherwise, if all the interaction is remote, a simple sequence of flows of UT of *actors* is sufficient.

4.3.1.3. Step 3 - Enterprise Structuring

Origin
The Enterprise Structure is the generalization of the original WISDOM software architecture, which was simplified in order to be readable and fit an adequate paper support (usually an A3 sheet) that allowed the representation of related Business Process (BP) Improvements (BPI). This generalization also originated the establishment of the business and software traceability by means of the User Tasks (UT) of the BPs and the WISDOM *tasks* and related interaction spaces generated from *essential use cases* decomposition. With the recurrent generalization of the WISDOM software architecture, the model started to be elaborated after the BP design, since at that stage enough information was already collected in order to understand which UTs were subject to which Business Rules (BR) and which Data Entities (DE) that they used, bot represented in the WISDOM software architecture, becoming the Enterprise Structure. This was the relevant business information available for the detail of *requirements* and software modeling in the Design phase.

Elaboration
The Enterprise Structure elaboration is the continuation of the BP design, and is based on the identification of the Business Rules (BRs) and Data Entities (DEs) with which the Interaction

Spaces (IX) must comply with. It involves the elaboration of the Interaction Spaces Diagram, based on the BP design, which later only needs rearrangement in order to be transformed in the Enterprise Structure. The diagram is the establishment of a relation of *business concepts* dependency in each IX, whether these are *regulations*, expressed by BRs, or *data*, expressed by DEs, its logical *resources*. The IXs are derived from the UTs relations, as one IX typically supports the *actor*'s interaction, representing a physical or computer supported space. In concrete terms, IXs represent offices, rooms, places as physical places, or computerized User Interfaces that allow both *data* management and the execution of programmed routines.

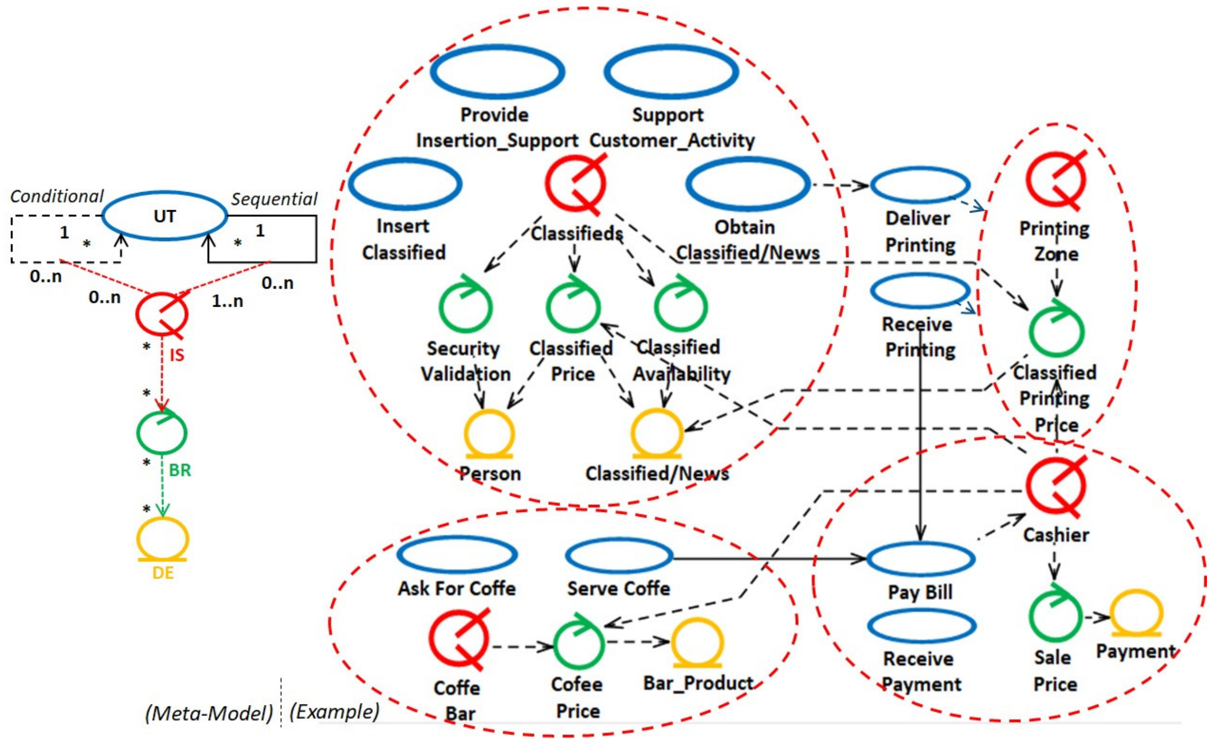


Figure 30. Enterprise Structure IX Diagram for "Publish Classified" BP.

The identification of IXs in the "Publish Classified" BP is presented in Figure 30, where it was defined that the four UTs of the BP are performed in the "Classifieds" IX, which must ensure the application of the BRs "Security Validation", "Classified Price", "Classified Availability" and "Classified Printing Price". The elaboration of the remaining IX is also presented, namely the "Printing Zone", the "Coffee Bar", and the "Cashier", each one with its BRs and DEs, namely for the prices of printing, coffee and final payment. At this stage, some of the dependencies are already established based on the semantic of each object e.g. the "Cashier" IX will use the "Classified Printing Price" (which relation is represented) and probably the "Coffee Price" (still not represented). In the final Enterprise Structure all relations that may be semantically be identified should be represented.

The Enterprise Structure elaboration follows, continuing the rearrangement of the previous diagram namely the BRs and DEs that should apply in each case, relating each component by means usage. The structure is representative of the social interaction support concerning the BP execution in terms of stable and essential norms, also known as the organizational kernel [Stamper, 2013]. The technique that uses the Interaction Spaces Diagram for IX identification and their relation to BRs and DEs is called as the GOALS *Enterprise Architectural Technique*.

Example

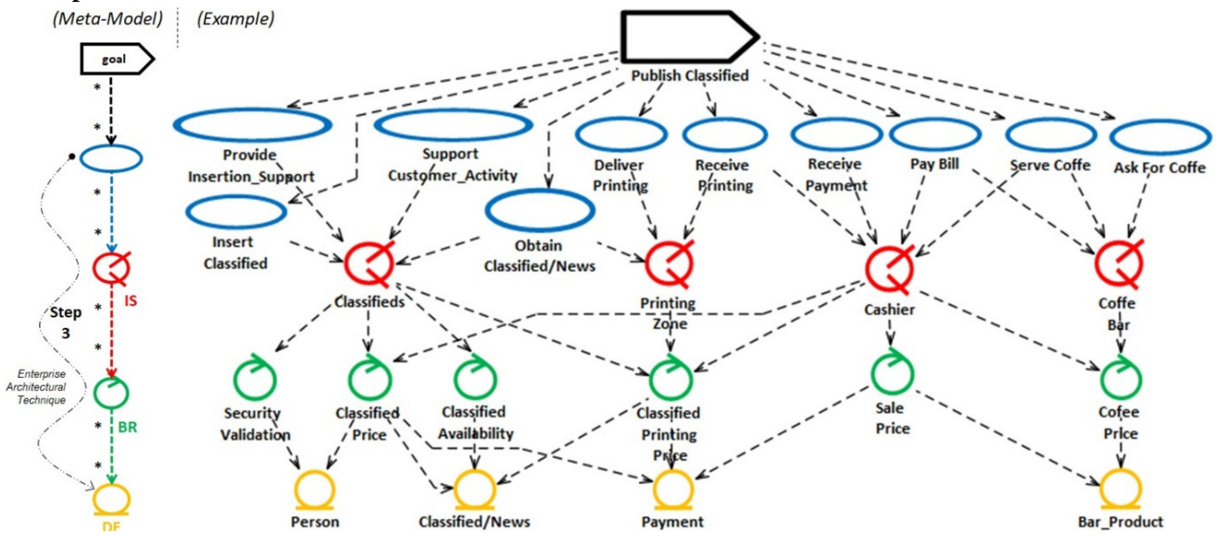


Figure 31. Step 3 - Enterprise Structure Meta-Model | Example of "Publish Classified" BP.

Figure 31 presents the Enterprise Structure of the "Publish Classified" BP presented in the Service Design, maintaining precisely the same structure of concepts that was elaborated in the Interaction Spaces Diagram. The structure of the Interaction Spaces Diagram for the "Publish Classified" BP is now related to the "Person" and "Classified/News") DEs by means of the affinity of the used *business concepts* semantics. A similar situation happens with the "News" IX, which is related to the "News Printing Price" and "Filter News" BRs, which uses the "Payment" and also the "Classified/News" DE, as presented in the Appendix C – "iKiosk" Project Diagrams, and the "Decoy BP".

Meaning to the SDP

The Enterprise Structure is the diagram that enables understanding the complexity of the ongoing BPI, and provides a structure of *requirements* to which the software development team must comply with in the process of developing the Information System. It is useful as a support to guide the elaboration of the detail of each of the UT in a last Step of the Analysis phase, and also for every Step of the software Design, since the IXs, BRs and DEs represent unchangeable information (unless erroneous) which can be interpreted as a persistent conceptual business structure that must be complemented with software specific parts, concerning their automation, from that moment on.

4.3.1.4. Step 4 - Task Modeling

Origin

The Task Model is originated from the need to understand what may be the best sequence of steps to accomplish the User Task (UT), according to a logical sequence of *data* management (of *business concepts*). The decomposition of the UT User Intentions using the CONCUR TASK TREES (CTT) method allowed the elaboration of a compromise between the user and the software development team that promoted cooperation, and allowed the structuring of the User Interface. By means of the application of the WISDOM *Software Architectural Technique* to each User Intention the internal structure of the system was also derived in terms of system responsibilities (components of the Business Logic), establishing the Task Model as a firm software development artefact.

Elaboration

The Task Model details the sequence of actions that leads to the completion of the UT under the *actor*' responsibility. It describes human behavior and is the first level of decomposition of the CTT method [Paternò, 1999], providing a second level of the structure of the User Interface by means of the Interaction Component (IC), following the specification of the Aggregation Space (AS) that supports the UT. From the UTs are derived User Intentions and System Responsibilities (SR) according to the decomposition of an *essential use case*. Each IC and supporting SR are named according to the meaning (the *business concept*, DE) which is being used. In each case, the name assumes an action (the user's responsibility) over the *business concept* (e.g. "Insert", "Delete"). The Task Model is modeled using an UML Activity Diagram (without the initial and final signs), specifying the flow of User Intentions, following the application of the *Software Architectural Technique*.

Example

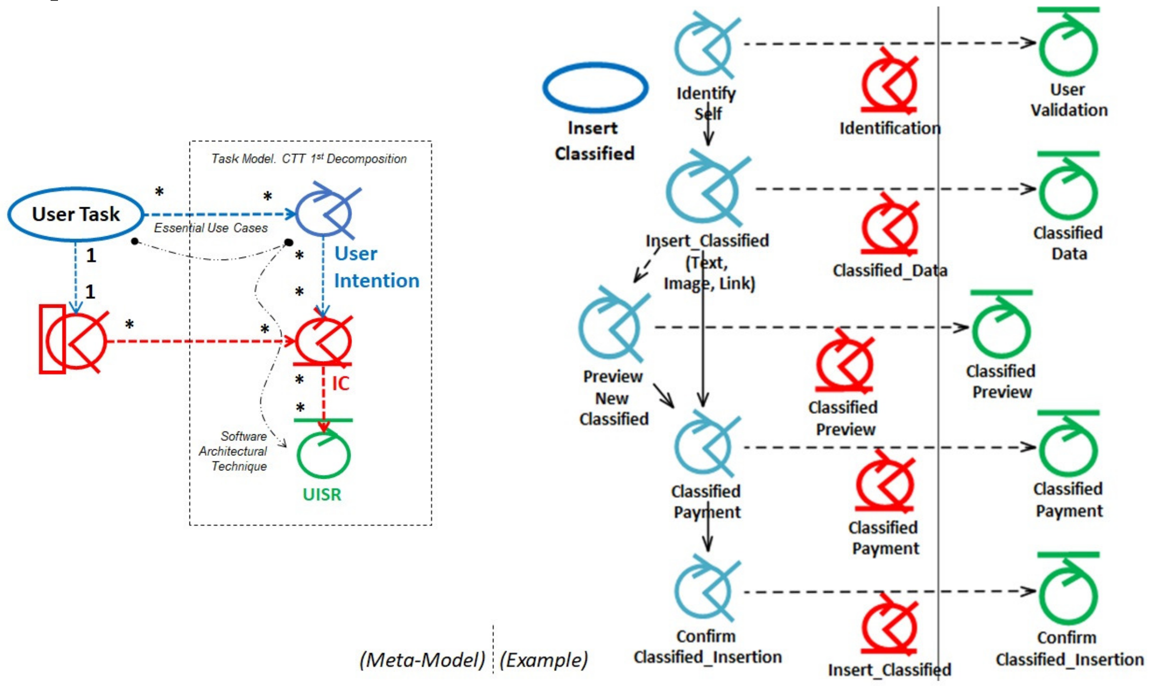


Figure 32. Step 4 - Task Modeling Meta-Model and Example for "Insert Classified" UT.

Figure 32 presents the meta-model of the Task Model, which defines that the AS is associated in a relation of 1 to 1 with the UT, and that the UT has many User Intentions. The Task Model, applies the *Software Architectural Technique*, eliciting ICs and UISRs (according to the defined interaction pattern). The example presents the Activity Diagram of User Intentions and its corresponding UISRs and ICs. The User Intentions "Identify Self", "Insert Classified (…)", "Previews New Classified", "Classified Payment", and "Confirm Classified Insertion", are supported by pairs of IC and UISR, namely: "Identification" IC and "User Validation" UISR, "Classified" IC and "Classified" UISR, "Preview" IC and "Return Preview" UISR, "Payment" IC and "Payment" UISR, "Insert Classified" IC and "Confirm Classified Insertion" UISR. For purposes of reuse of the software-specific components, the managed *business concepts* are compared to the DEs already existing in the Domain Model (for purposes of validation). *Business concepts* are nouns and the intentions are the verbs of the User Intentions. If a new *business concept* is identified, then it must be clarified if its meaning is already included in the existing DEs (e.g. "Classified Preview" in "News/Classified" DE) or included as a new DE.

Meaning to the SDP

The Task Model details User Tasks (UT) in order to obtain information to carry on the User Interface design, which happens in Step 7 – User Interface Design, and it defines the perspective of a specific *actor* concerning an Interaction Space (IX), which has a direct one-to-one relation with the software-specific component, the Aggregation Space. It represents the identification of the tools which are used along the UT, which may be implemented by the Information System or not, to which the relation of: the UT with the AS; and of the User Intentions with the SRs; can already be established in this last step of the Analysis phase, prospecting what will be probably implemented. In the case when new Data Entities are identified then they must also be expressed as DEs, that also update the Service Design. This DE identification process, should be maintained up to the Step 7 (presented in Section 4.3.2.3. Step 7 – Database Structuring), where the DEs structure will be detailed in terms of tables and fields of a Database.

4.3.2. Design Phase

The Design phase is the continuation of the method, gradually transposing the responsibility of elaboration from the business to the software domain. The Enterprise Structure of the ongoing Business Process Improvement (BPI) is the reference model that provides the *requirements* for the production of a Software Architecture for in-house software development. The process continues in a top-down logic, by means of detailing the User Intentions of each User Task (UT) in User Interactions (Step 5 – User Interface Design) aiming the design the User Interface, the Business Logic (Step 6 – Business Logic Structuring) and the Database (Step 7 – Database Structuring) structures, getting gradually closer to a level of detail that suits implementation, and finishes with the elaboration of the Software Architecture (Step 8 – Software Architecting).

4.3.2.1. Step 5 – User Interface Design

Origin

This step is originated from the need to design the User Interface in a top-down process in order to ensure that all the needed information for this purpose was elicited from business *requirements*, and that the most convenient technological solutions would not influence the system being designed. With the application of the CONCUR TASK TRESS (CTT) method, and the structuring of User Interfaces with the Hydra Framework [Costa et al., 2007], which supports the Model-Driven implementation of the WISDOM interaction space (now referenced as Interaction Components (IC)), the detail of the User Interface fields becomes the remaining concern. The detail of the IC and its Fields is useful for the Business Logic and Database structuring, placing the User Interface Design as the driving model of the system specification. With the formalization of the GOALS method, the BEHAVIOR DRIVEN DEVELOPMENT (BDD) method was used to articulate the dialogue of the User Interface by means of the elaboration of an Interaction Model which is complemented with the Canonical Abstract Prototypes (CAP) [Constantine et al., 2003], for the final design of the User Interface.

Elaboration

The Interaction Modeling specifies each User Intention in terms of User Interactions. It also defines Aggregation Space (AS) usage and the behavior of the system in terms of the System Responsibilities (SR) triggered by Interaction Objects (IO), which are the components with which the user interacts with in terms of e.g. clicks and keyboard inputs. It is used to generate a User Interface design, part of the Business Logic, and identify the Fields of the Data Entities (DE). The Interaction Model is carried out by means of the application of the BEHAVIOR DRIVEN DEVELOPMENT (BDD) method [Chelimsky et al., 2010]. BDD is an agile software development process that describes the system behavior based on a User-Centered Design perspective, producing pseudo-code for User Interface specification by means of the elaboration of User Stories. User Stories describe a feature for the support of a certain purpose (the UT) within a certain scenario which in GOALS is specified by a state of the system within a certain AS. BDD's pseudo-code has the following syntax.

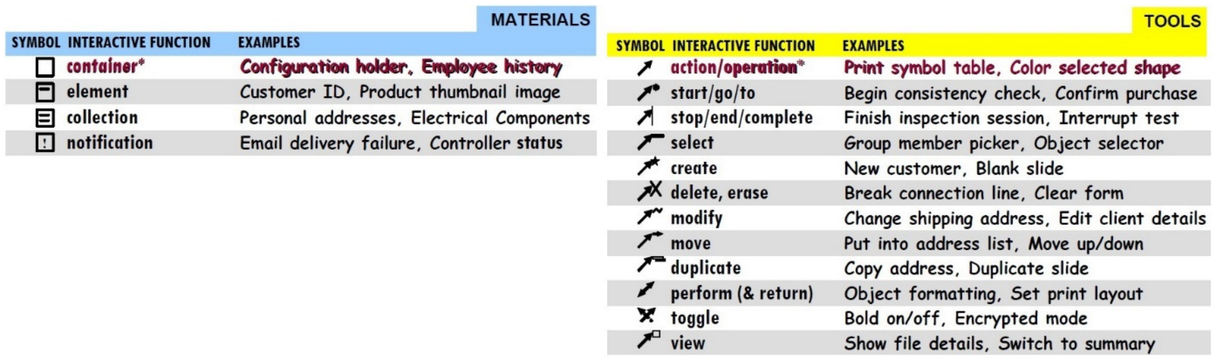
Given [State] **When** [Interaction] **Then** [System Behavior]

Where [State] represents the actual state of the system given the information of the AS with which the user is interacting with; [Interaction] is a flow of User Interactions that specify User Intentions; and [System Behavior] is the expected outcome that triggers User Interface (UISR) and Database (DBSR) SRs. The software architectural components are derived from BDD pseudo-code by means of the relation presented in Table 7.

Table 7. BDD Pseudo-Code and GOALS Components Relation.

BDD pseudo-code	GOALS Component
Given	Aggregation Space (identification) + [State]
When	User Interaction + Interaction Object + SR
Click, Choose, Set, Type	User Interactions 'Click', 'Choose', 'Set' or 'Type'
Display 'Page' or Go to 'Page'	User Interface SR 'Display Page' + IC 'Page'
Field	Data Entity Field
Then	(last) System Responsibility (SR)

The Interaction Model defines that a User Story is applied to each UT, composing a User Interface that references Interaction Components (IC) and Interaction Objects (IO) that together compose the AS of that UT. The User Story starts with a **Given** clause that identifies the AS and its [State] if necessary, and specifies a **When** sentence for each User Intention (of the Task Model) identifying new IOs and SRs by means of a second application of the WISDOM *Software Architectural Technique*, finishing with behavior specification in the **Then** clause.



MATERIALS		
SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
☐	container*	Configuration holder, Employee history
☐	element	Customer ID, Product thumbnail image
☰	collection	Personal addresses, Electrical Components
!	notification	Email delivery failure, Controller status

TOOLS		
SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
↗	action/operation*	Print symbol table, Color selected shape
↗	start/go/to	Begin consistency check, Confirm purchase
↗	stop/end/complete	Finish inspection session, Interrupt test
↗	select	Group member picker, Object selector
↗	create	New customer, Blank slide
↗	delete, erase	Break connection line, Clear form
↗	modify	Change shipping address, Edit client details
↗	move	Put into address list, Move up/down
↗	duplicate	Copy address, Duplicate slide
↗	perform (& return)	Object formatting, Set print layout
✕	toggle	Bold on/off, Encrypted mode
↗	view	Show file details, Switch to summary

Figure 33. Canonical Abstract Prototypes (CAPs).

Each IO is then customized according the CANONICAL ABSTRACT PROTOTYPES (CAPs) [Constantine et al., 2003] and its compatibility with the BDD operations, namely to support: BDD "Type" with the "Modify", "Select" with "Select", and "Click" with the "Start/Go/To" CAP, while the space that the IO occupies is expressed by the space of the CAP container in the User Interface design. The CAPs are presented in Figure 33.

Example

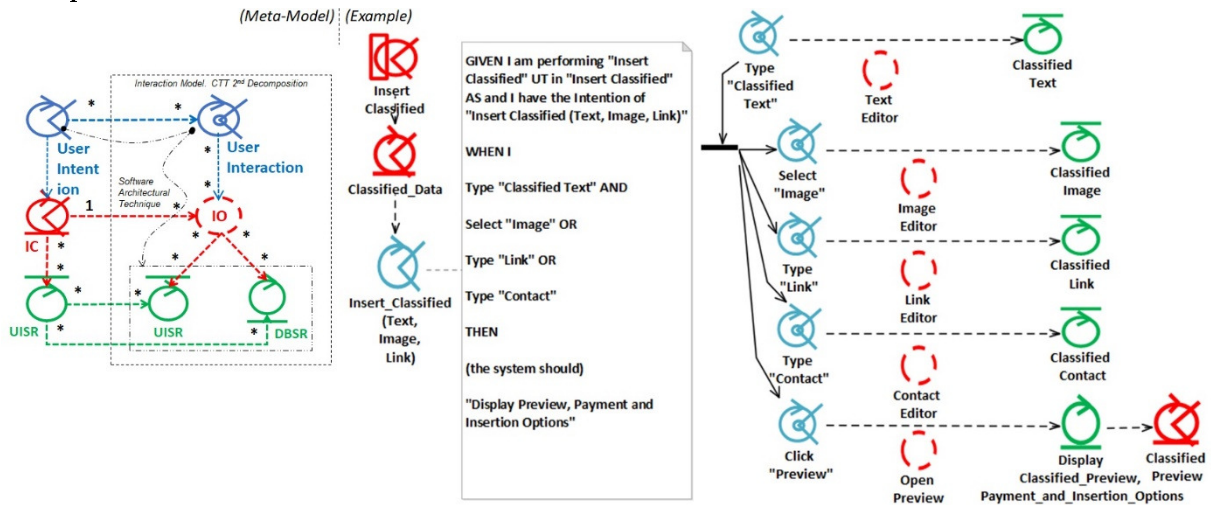


Figure 34. Step 5 - Interaction Model Meta-Model | "Insert Classified (Text/Image/Link)".

Figure 34 presents the Interaction Model meta-model that states that a User Interaction is supported by an IO that uses UISRs and DBSRs. The example presents the decomposition of the "Insert Classified (Text, Image, Link)" User Intention into the "Type Classified Text", "Select Image", "Type Link", "Type Contact" and "Click Preview" User Interactions.

These User Interactions, similarly to the Information System structuring which was carried out in the Task Model by means of the software architectural technique, have a correspondence, now with IOs instead of ICs, which in the same way are related to SRs, the first ones UISRs, and the last one the "Display Preview, Payment and Insertion Options" DBSR, which is a convention concerning the application of the BDD method to implement the chosen interaction pattern, towards the User Interface and Information System structuring.

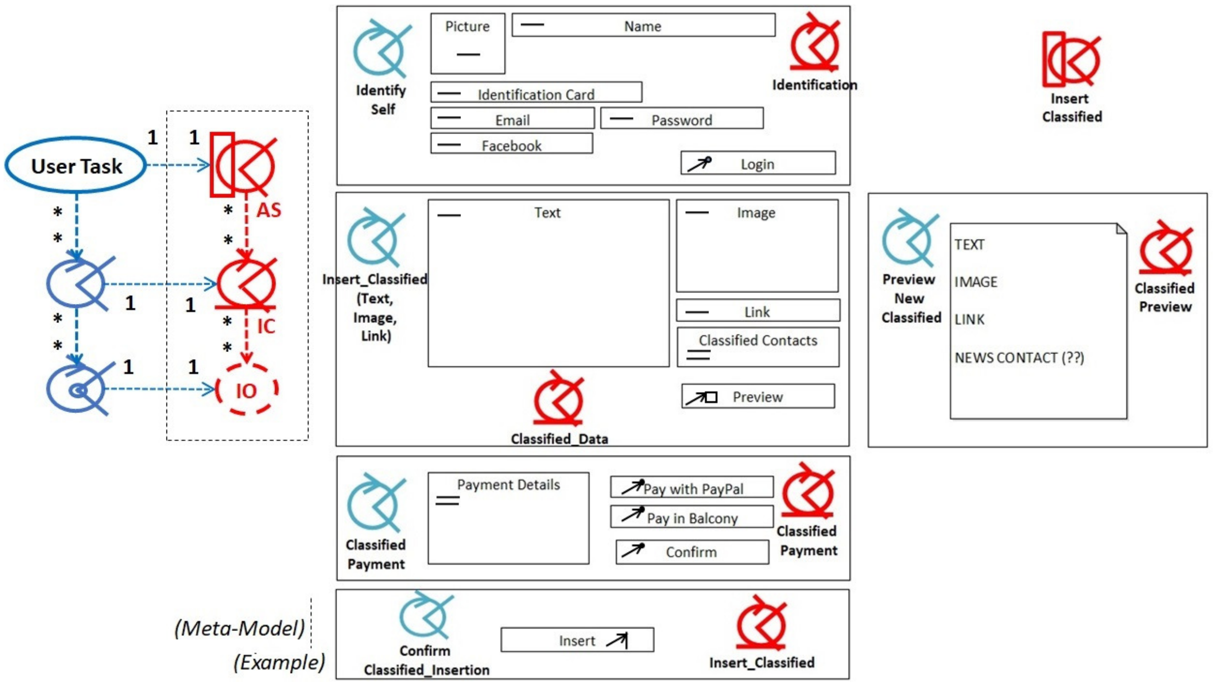


Figure 35. Step 5 - User Interface Design of "Insert Classified" AS.

Figure 35 example the Aggregation Space “Insert Classified” User Interface design, which was developed for the iKiosk project. It is composed of five ICs, as identified in the UT “Insert Classified” in terms of User Intentions, and each IC is composed by its Interaction Objects as a result of the elaboration of the Interaction Model for each User Intention. It includes customer personal *data*, enables the introduction of a classified, its previews, presents the payment in case it applies, and has a final “Insert” button.

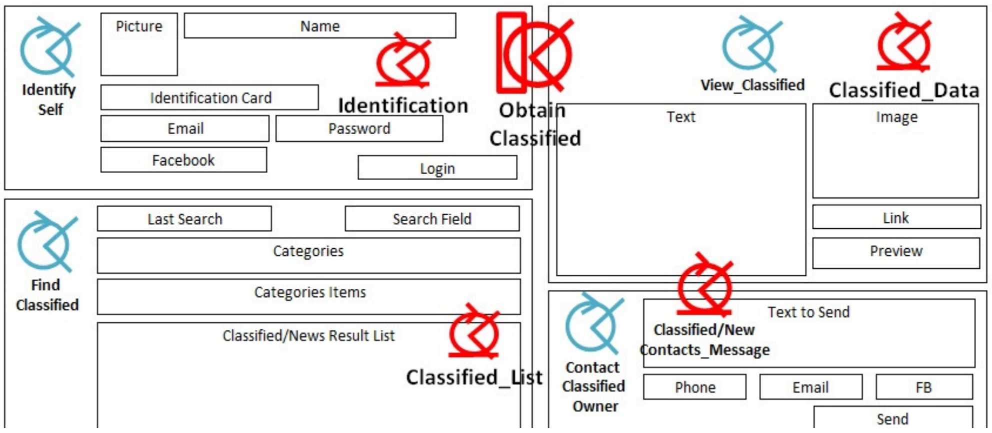


Figure 36. Example of User Interface Components Reuse.

We take as example the “Classified Data” IC in order to illustrate the reuse of ICs which our method facilitates. The IC is composed of the Fields “Text”, “Image”, “Link”, “Classified Contacts”, and “Preview” which makes the navigation to the “Classified Preview” IC. Since a new User Intention, “View Classified” (in the original “Obtain Classified” UT), was identified as using the same IOs, then the “Classified Data” IC was reused concerning the “Obtain Classified” AS, which supports the UT with the same name. According to the Interaction Model, the behavior of both “Insert Classified” and “Obtain Classified” AS, may change.

Website Design

In order to elaborate the website design, the information from the “News/Classified” and “Advertisement” DEs had to be presented to the user in order to fulfill the objective of establishing a “decoy” that would attract people to the “iKiosk” daily information. As the related DEs were produced in the “Publish Classified”, “Publish News” and “Publish Advertisement” BPs, the UTs that produced this information were analyzed targeting software-specific components reuse for the support of the “Obtain News” UT of the “News Decoy” BP. Thus, to the original “Obtain News” UT decomposition in User Intentions, new User Intentions were identified which were related to the “Classifieds” and the “Advertisements”, as the user could also be interested in business opportunities from sponsors or other “iKiosk” users (e.g. part-time jobs or car selling). This lead to the unification of the “News” and “Classified” software structure, as both structures have the same common entities related, namely the “Categories” and “Items”, the detail (Text/Image/Link), the Contacts, the way of obtaining it (by Phone, eMail or Facebook), and the payment, to which the “Advertisements” were added composing the website front-end. The website allows the search of news and classifieds using the same tools, producing a common list that may be filtered in order to display only one or the other, from which one may be chosen, so that the user may take action contacting of obtaining the item according to the available options. The list of advertisements would be filtered according to the terms search and the level of promotion which was accorded concerning each advertisement.

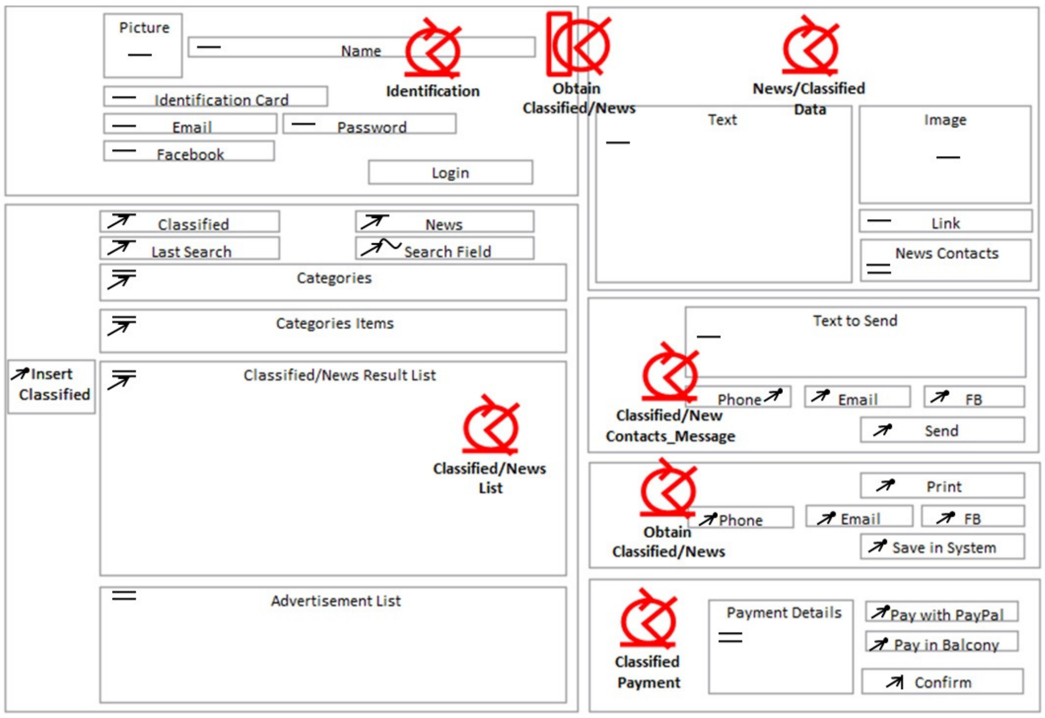


Figure 37. User Interface Design for the Front-End of the "iKiosk" website.

In the User Interface presented in Figure 37, it is possible to identify the reuse of components by means of comparison with the "Insert Classified" AS, to which are added the "Advertisement" IC, and the generalization of the "News" and "Classified" structure which promotes software simplification. This is reflected in the Software Architecture in terms of the generalization of the "News" IC, that now supports both classifieds and news, and for that reason had its name changed to "News/Classified", which Business Logic and Database components are the same than the "Classified" IC, invoked with distinct parameters by the invoking AS, "News" (the website) and "Classified", respectively. A User Intention of "Insert Classified" was added to the "Obtain News" UT, from which the "Insert Classified" User Interaction was added, establishing the navigation from the "Obtain News/Classified" AS to the "Insert Classified" IC, which is reflected in the Software Architecture by a usage relation from the first to the second. As a complementary result of the unification of the "Insert Classified" and "Obtain Classified/News" UTs, the web site could also be used in the "iKiosk" store in order simplify the "Insert Classified" BP, in which case, the "Customer" actor and the "Attendant", would only perform a single UT in the "Classifieds" IX.

Meaning to the SDP

Once the User Interface design is finished, the *User Interface Designer* will begin to realize what are the more complex parts of the Information System, so that he may focus the software development efforts in ensuring that these parts will be properly elaborated. Moreover, by eliciting the fields of the User Interface, and its behavior, the Software Developers can program the components under their responsibility with a higher level of knowledge concerning the desired functioning. This knowledge may have a positive impact in the software development performance, by ensuring that the programmer knows what is desired in terms of User Interface appearance, arrangement, desired sequence of fields and commands usage, the Business Logic structure expected behavior and the Database tables and fields structure. It is what drives the design and the development of the Information System in terms of its detail.

4.3.2.2. Step 6 – Business Logic Structuring

Origin

The Business Logic structuring became the logical sequence of the elaboration of the User Interface since based on the application of the *Software Architectural Technique*, the Business Logic System Responsibilities (SR) are already identified and need to be organized in terms of Database dependency. As the model also provides information for the Database structuring namely in terms of used fields, it is therefore an essential step of the Information System structuring, and the beginning of the Software Architecture elaboration.

Elaboration

The Business Rules (BR) are the foundation of the Business Logic as the business-specific SR of the model, which are now complemented with the User Interface (UISR) and Database (DBSR) SRs. SRs and BRs need adjustment in terms of Data Entity (DE) usage, which is carried on based on the used Fields identified in User Interface design. First it is necessary to define the DEs to which each Field belongs to, establishing a relation between the BRs and SRs, and the DEs. The relation between SR and the User Interface related components: Aggregation Space, Interaction Space, Interaction Components and Interaction Objects; can also be represented for the purpose of analysis of this first approach to the Software Architecture, and for clarification of the Business Logic components usage. At this stage, the *Software Architect* is already aware of the existing *business concepts*, since these have been identified and elaborated since the Service Design (Step 1) of the Analysis phase. Once all the identified User Interface Fields are properly supported by the Business Logic components as expected in the User Stories, the Business Logic structuring can be considered as finished.

Example

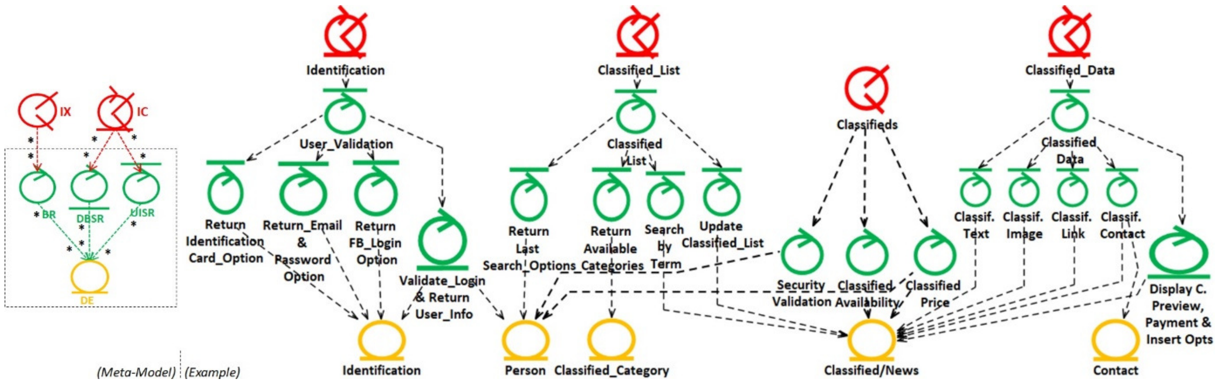


Figure 38. Step 6 - Business Logic Structuring Meta-Model and Example.

Figure 38 presents the meta-model of the Business Logic structuring, which states that the Information System has many BRs, the IC and the IO have many UISRs and DBSRs, reflecting the "Read and Save – Interaction Design" pattern. The example presents a structure of SRs derived by means of the detail of User Intentions, generating the "Classified" and "News" UISRs, and the detail of User Intentions in User Interactions, generating the "Text Options", "Image Options", "Link Options", and "Contact Options" UISRs, and the "Make Available Preview, Payment and Insertion Option" BDSR. Both "Classified" and "News" UISRs use the same SRs due to a generalization of the "News" and "Classified" concepts that led to the specification of the "Classified/News" concept, since it was identified both were consistently related to every other concept in a similar way, and distinction would be made by means of the type of record and the parametrization that would apply to each.

Meaning to the SDP
Once the Business Logic structuring is finished the software design process can be considered as "almost" finished in terms of its design. The remaining modeling parts are simply the finish of the establishment of the relation that were defined after the Enterprise Structure elaboration. The method follows to the Database structuring and final Software Architecture.

4.3.2.3. Step 7 – Database Structuring

Origin
The Database structuring is the core software engineering model, since it is where the *data* of the system is persistent over time, and where no structural errors may exist without affecting negatively the Information System functioning. For this reason, the Data Entities relation elaboration starts on the first Step of the Analysis Phase, and its detail in terms of Fields is also carried out along the remaining SDP ensuring that this model is always updated when new relevant information is elicited. The Database structuring represents the base of the conceptual business and software structures.

Elaboration
Data Entities (DE) identification started in Step 1 - Service Design, and continued along the SDP, so at this stage the practitioner already has a well-defined notion of the *business concepts* under analysis. In terms of Database objects, DEs are implemented by Tables, and its attributes by Fields. DEs are related by means of syntactic relations with multiplicity of typically one-to-many, or many-to-many. DEs relations define restrictions that are not subject to the logic of design specifications of the User Interface and the Business Logic in which they are used. Here, the Business Concept is worked on its full scope i.e. if one DE can directly relate "in any means" to another, a relation of one-to-many is established from the DE that uses, to the DE that is used. Complementarily, the Fields identified in the previous Steps, are also associated to its containing DE. The Database Structuring is now possible once all DEs and DE' Fields are identified.

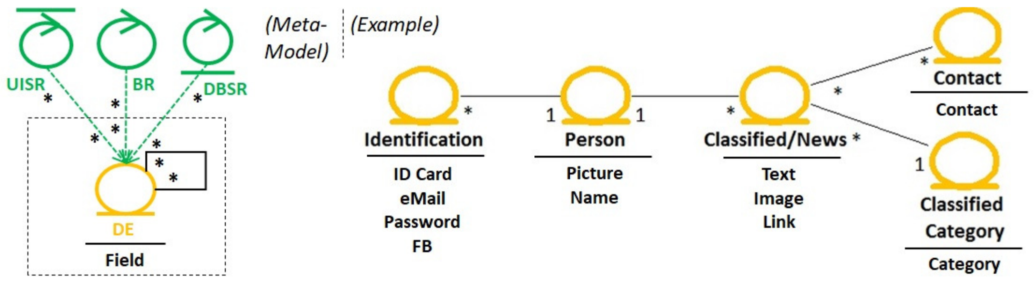
Example


Figure 39. Step 7 - Database Structuring Meta-Model and Example.

Figure 39 presents the meta-model for the Database structuring, which defines that the DE can be used by any of the Business Logic components, the BR, UISR and DBSR, in a relation of many to many. The example (partially) presents the Database specification, reflecting a situation where DE "Person" is related the "Classified/News", meaning that a customer can publish many "Classifieds" or "News", since both the "Reporter" and "Customer" or other user interacting with the system are represented in the "Person" DE. The DE "Payment" has a relation of many to many to the "Classified/News" DE since a payment can include more than one classified or news, and each of these may also have more than one payment. In a complete representation of the Database, all the fields identified in the User Interface must be present.

Meaning to the SDP

The Database if the core element of the Information System, and for that reason, and since the system was designed in top-down process, the remaining Step is the Software Architecture elaboration, which precedes implementation.

4.3.2.4. Step 8 – Software Architecting

Origin

The Software Architecture as presented by GOALS is an evolution of the WISDOM architecture, with the inclusion of the Enterprise Structure, which logic of dependency is also originated from itself. In other words, the Software Architecture is an evolution of itself, after a process of generalization, also including the original WISDOM Architecture. The Software Architecture is the model that allows the distribution of software development work targeting development performance.

Elaboration

The Software Architecture relates all the components identified in the previous Steps (except for the User Intentions and User Interaction classes) in a single structure by means of a relation of usage. The Software Architecture can be used to specify priority and allocation for a software development team. Priority logic is contrary to the logic of components usage. Hence, components should be implemented from bottom to up, as the upper objects use bottom ones.

Figure 40 presents the Software Architecture, with the relations previously introduced in the language meta-model, in a top-down relation from the User Interface, the Business Logic and the Database components, where the Enterprise Structure relation are highlighted with a stronger line. Each AS supports a UT with the same name, having therefore a relation with the Enterprise Structure by means of the UT and the used IX. The original "Obtain Classified" AS is represented to highlight the reuse of the ICs and underlying UISRs, DBSRs and DEs.

Example

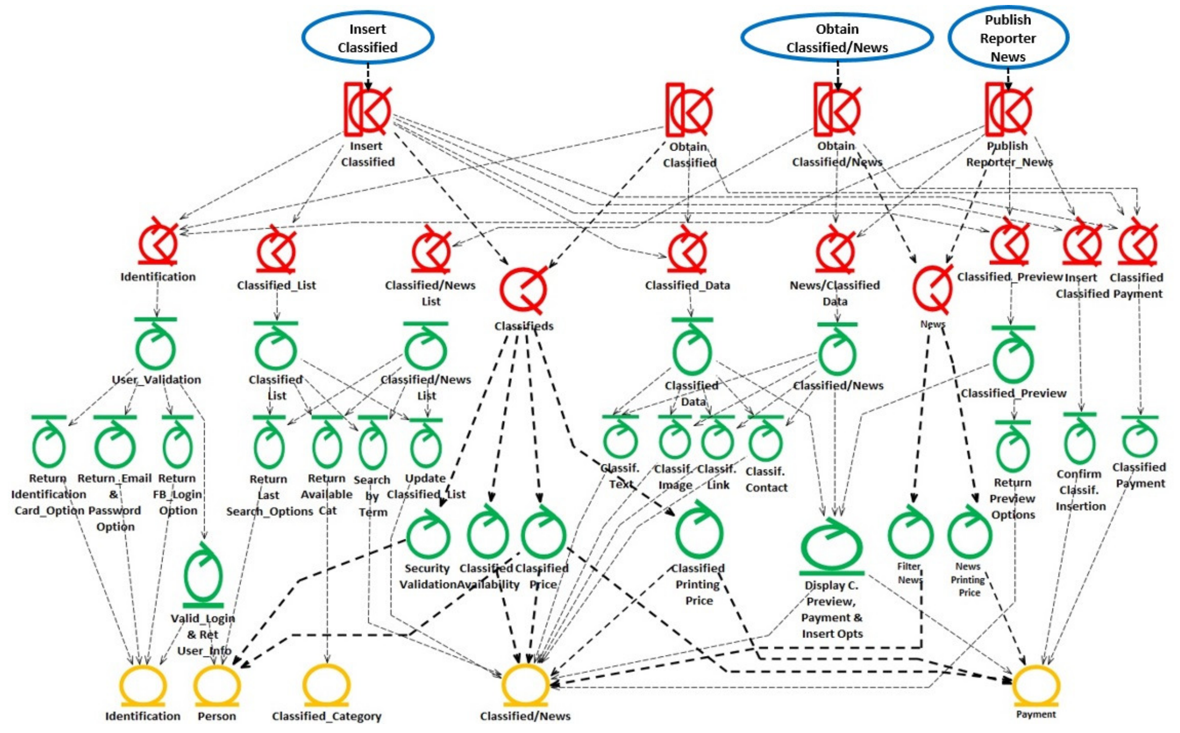


Figure 40. Step 8 - Software Architecture Example.

Meaning to the SDP

The Software Architecture is the more important diagram that should be pursued along the SDP, as it represents the solution for the BPI problem, and it also allows the specification of the best possible organization of the software development team, by means of specifying the precedence of implementation of its components, contributing to promote the SDP performance, which continues with the implementation of the Information System.

4.3.3. Case Study Conclusions

Concerning the GOALS method application, the "iKiosk" project served to present the concepts used in each model, and how the modeler may develop, always in a "top-down" sequence, the enterprise business and software models. The applied techniques follow the meaning of each (previously presented) concept, maintaining the coherence of the produced logical structure as the modeling of the BPI is carried on, maintaining the coherence between the complexity of the problem and the complexity of the solution. The complexity of the problem is reflected by the human activity to which an automated supported is needed, and of the solution depends on the corresponding software architecture in terms of User Interfaces (Aggregation Spaces), Interaction Components and Interaction Objects, and their supporting Business Logic.

Table 8. "iKiosk" Project Development Process.

#	Date	Stakeholders	Phase	Steps	H
1	07-02-2017	*Business Owner* *Software Architect*	Analysis	1. Service Design	4
2	12-02-2017	*Software Architect*	Analysis	1. Service Design 2. BP Design	5
3	21-02-2017	*Business Owner* *Software Architect*	Analysis	2. BP Design 3. Enterprise Structure	4
4	05-03-2017	*Software Architect*	Analysis & Design	3. Enterprise Structure 4. Task Model 5. User Interface	5
5	18-03-2016	*Software Architect* *User Interface Designer*	Design	4. Task Model 5. User Interface 6. Business Logic 7. Database	15
6	26-03-2017	*Software Architect* *User Interface Designer*	Design	4. Task Model 5. User Interface *Conciliation of Classifieds and News Architecture* 8. Software Architecture	15
7	14-04-2017	*Business Owner* *Software Architect*	Analysis & Design	*Project Validation*	1
					49

The "iKiosk" project had the production moments presented in Table 8, in which the intervenient stakeholders are distinguished. The *Business Owner* whom accumulated the roles of *Service Designer* and *Business Process Designer* participated in three moments (meetings). The *Enterprise Architect* assumed the Analysis of the business, and was joined by a *User Interface Designer* in the Design phase for the User Interface specification. The total number of hours is presented in column "H", in a total of 49 for the complete project specification. The project is currently in the process of acquiring the *resources* for its implementation.

4.4. Case Study - "Star Project"

We now present a case study of the application of GOALS that includes the organization of the business of a new enterprise and all the Method Steps that the analyst should go through before reaching a stage of software implementation. The "Star" project is a real software project that has the implicit intention of creating an enterprise, and the explicit intention of creating an Information System for the World Wide Web, targeting large audiences of people that want to support charity. We present the project as follows.

The "Star Project" was developed for a local Buddhist community in Madeira, Portugal, which aims at raising funds to invest in areas of the globe which have basic chasms e.g. water and food supply, namely in Africa and Asia. The project is named as the "Star", in allusion to the will of creating a patterned process of society development that could be replicated in case of success "illuminating" a different road for success of poor countries and cities. The objective is to create an enterprise that would channelize investments from people that would be interested in helping other people located in remote places.

The idea is to elaborate the project according to the Buddhist philosophy of reaching an independence, in this case for the people that promotes the investments and its society, as long as those investments are considered as fruitful for the location, and as long as they promote structural changes that benefit the community and fill relevant chasms. The project had as primary objective the creation of a website that promoted the creation of a support community for investment projects, that may receive supporter feedback and pre-investments, as presented by promotors at existing investment capturing sites (e.g. Kiva [Kiva, 2017] or Zidisha [Zidisha, 2017]), to which the "Star Project" site would channelize investment.

Since it was predicted that the enterprise once fully operational would have an indirect relation with considerable amounts of money, and would also involve the creation of full-time jobs, a strategy was defined, first targeting the development of the minimal software implementation possible, and also the minimal involvement possible of the *Business Owners* and colleagues in terms of spent time. Thus, the strategy was to minimize investment, aim the development of a full-operational pilot project which once found successful, would posteriorly be increased in terms of people, business and software development in a second project. The roles involved in the project were two *Business Owners*, a *Software Architect* and a *User Interface Designer*.

The project predicted the execution of two phases:

- **Business Modeling**. Given the complexity of the project, distinct iterations of the Analysis of the business were defined. In each iteration the produced models were validated until there was a solid idea of the processes that would support the business in a regular basis, so that it would be possible to identify all the *tasks* that would support the website information production.
- **Website Modeling**. Once the full complexity of the business was acknowledged, namely identifying the most basic concepts that would support it, the web site design would take place, so that it would also be possible to estimate the software development effort and consider the best implementation options, considering the possibility of applying for European funds, concerning its implementation.

The following GOALS models were produced:

- 1. Service Design
- 2. Business Process Design
    - "Grow Community"
    - "Support Project"
    - "Capture Investment"
    - "Evaluate Investment Support"
- 3. Enterprise Structure
- 4. Task Model
    - User Tasks of "Grow Community" BP
    - User Tasks of "Support Project" BP
- 5. User Interface Design
    - "Attract Project Community" of "Grow Community" BP
    - "Support Project" of "Grow Community" BP
- 6. Business Logic Structuring
- 7. Database Structuring
- 8. Software Architecture

We present the totality of the produced diagrams in order to illustrate our language and method, as follows, and the remaining diagrams are presented in Appendix D – "Star Project" Diagrams.

4.4.1. Step 1 - Service Design

The design of the service went through a phase of inception in order to understand how could be produced the project information for the customer (supporter in the project perspective). We defined the *ideology* of the project, based on the High-Level Concept [Kreitzberg, 1996], as: "Capture of Charity Investment to promote the Sustainability of Places with Chasms". In order to specify the Production Logic of the enterprise, we started by elaborating a Domain Model of the logical structure that consumes and produces the *business concepts* (Data Entities (DE)). After, we followed to the organization of the Business Processes (BP) that would manage those DEs, and following that, we identified the *actors* of the enterprise that would assume the responsibility of initializing and participating in each BP. Finally, we defined the lines of interaction and visibility, separating the internal and external *actors*.

Data Entities Identification

The following text describes the Domain Model and justifies existing choices. The promoters (DE "Promoter") and their investment (DE "Investment") proposals at fund capturing sites where considered strategical as sources of opportunity to help needing countries (DE "Place") and specific regions in basic issues such as water, food, education and environment (DE "Chasms"). However, the idea also targeted the edification of benevolent permanent changes (DE "Benefits") that would promote the filling of the region chasms gradually and consecutively, aiding its development process. The idea was to channelize as much investment as possible, not towards too many investments, but to strategic investments that would fill existing chasms, thus, the investments could be more than one towards the same benefit, reason why are considered investment channels (DE "Investment Channel"), which cross multiple investments to produce a synergy (DE "Synergy") for the region. A synergy is an added value from two or more efforts that generates a total which is more than the sum of the parts, i.e. building a bridge and a road may add more benefit than the road and the bridge independently, as it may promote the region commerce and its sustainability (DE "Sustainability").

Business Process Identification

Four BPs where were conceived in order to organize the activity of the enterprise. Considering the project implementation strategy, the focus was on the “Grow Community Project” BP, as the process that would produce the information to present in the website driven by the process of elaborating the projects based on the “Capture Investment” BP. The “Grow Community Project” BP would capture investment intentions, present projects, and grow the community at the same time, creating a logic of (customer) support (“Project Support” BP) that started before the investments, would accompany the project along the its evaluation (“Evaluate Investment Success” BP), and would only finish after the project end.

Actors Identification

Figure 41 presents the Service Design. Four *actors* were identified and related to the BPs in which they participated or initiated. The “Supporter” as the external *actor* (placed outside the red line of interaction), the “Mission Manager”, “Project Manager”, and “Project Builder”. From a specific relation of the DEs, was elicited the main *business concept* structured logic of the project. It is (highlighted with a stronger line of association) the elaboration of “Synergies” of “Investment” which will produce “Sustainability” based on its benefits, which are measured according the defined and readable “Success Parameters”.

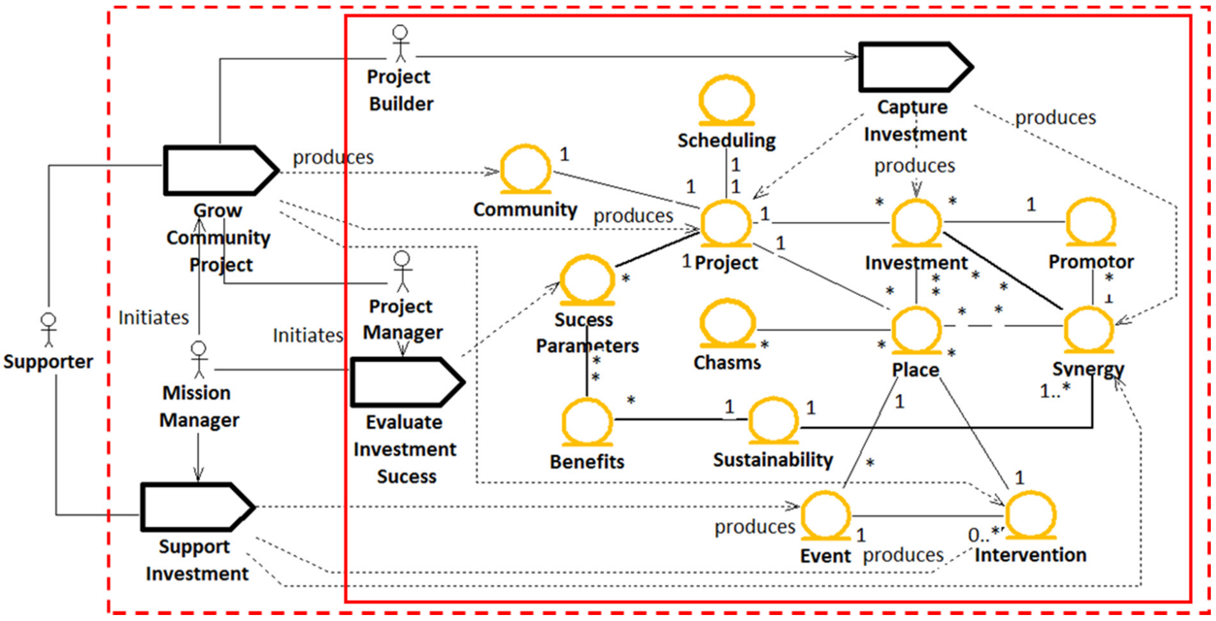


Figure 41. Step 1 - Service Design (“Star Project”).

By the analysis of the Service Design it is important to acknowledge the produced and consumed DEs, namely the ones that reflect the axis of the project. The “Project” DE is produced by the “Grow Community Project” BP, but only after the “Investment” and “Synergy” DEs are produced by the “Capture Investment” BP. The axis DEs are considered to be produced along with the “Project” and “Synergy” DEs. The more relevant relations of production are: the “Capture Investment” BP and the “Investment” and “Synergy” DEs; the “Grow Community Project” BP and the “Community” and “Project” DEs; the “Evaluate Investment Success” BP and the usage of “Success Parameters” DE; and the “Support Investment” BP and the “Event” and “Intervention” DEs. These relations are the top and the bottom of a relation of concepts that is structured upon the semiotic logic of *business concepts*, their *syntactic, paradigmatic* and *syntagmatic* relations that will compose the *knowledge* of the Information System, to suit the *ideology* of the enterprise.

4.4.2. Step 2 - Business Process Design

The four Business Processes (BP) were designed, and the focus was directed to the "Grow Community Project" BP which would implement the logic of obtaining supporter to the ongoing projects by means of the user interaction with the website, also providing possibilities of participation. The design was carried out based on the *Cooperation Logic*, which defines that the BP goes from one *actor* to the other by means of complete UTs using logical Interaction Spaces, as all the conception of the project was to be based on remote interaction except for the one predicted in the "Support Investment" BP, which concerns the preparation of support events and interventions in the place. We now present the logic with which was produced the "Grow Community Project" BP.

User Tasks Identification

The "Grow Community Project" is the BP which produces information to feed the website in order to capture supporter for existing investments. The idea is to obtain support from the customers, as the "Project Manager" presents the approved projects, which represent strategical investments by means of the "Present Project" UT. Once the "Project Manager" presents the project, the "Present Project Activity" UT, carried out by the "Mission Manager" Actor follows, in order to increasingly create a project community that would be interested in participating in distinct support possibilities, namely, investments, in-loco interventions, deals or donations.

Interaction Spaces Identification

The identification of Interaction Spaces (IX) is done in a patterned way, knowing in advance that every time two *actors* transmit information from one to the other, by executing one UT and the other, and that one IX will be used for that effect. In the case of the "Grow Community Project" BP, IXs were named after the more important Data Entity (DE) that characterized the *transaction*, or the logic of cooperation. These IXs where the "World Forum", reflecting the wide "space" which is created by presenting to the "World" the projects that need support, and the "Project", which is characterized by the project information that will be presented. As all the human interaction is remote, the BP was presented using a simple flow of UTs, without the representation of the physical space of each IX.

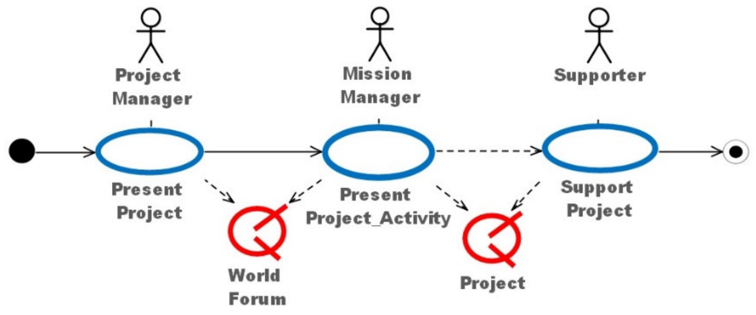


Figure 42. Step 2 – Business Process Design of "Grow Community Project".

The "Grow Community Project" BP is presented in Figure 42. The BP flow reflects the need to capture the Actor "Supporter" support, to which the "Present Project" and "Present Project Activity" UTs are strategical. As the "Supporter" performs his "Investment Feedback" UT in the "Conceive Investment Project" BP, the "Project Manager" may monitor the feedback until he feels that has the conditions to present the project by means of "Present Project" UT. Once the project is presented, the Actor "Supporter" may execute the "Support Project" UT as many times as he may wish.

4.4.3. Step 3 - Enterprise Structuring

The Enterprise Structure was carried out by means of the elaboration of an Interaction Spaces Diagram that related Business Rules (BR) to the identified Interaction Spaces (IX) in order to understand the main concerns when dealing with the information present in each of the UTs that associated to each IX. After, the Enterprise Structure was finished, by means of relating each BR to the DEs identified in Step 1 - Service Design.

Business Rules Identification

The elaboration of the Enterprise Structure was oriented by means of the identification of the concerns and *regulations* that should apply in each of the identified Interaction Spaces (IX). The main concern is the law of existing countries, and the application of the principles of the project, namely regarding the creation of synergies of channels of investment, and the defense against false promoters and projects. For each IX, the Business Rules (BR) that should be applied were identified: “Investment Applicable Law”, “Expected Benefits & Population ROI”, “Synergy Specification”, “Sustainability Warranties” and “Intervention Viability”; are examples of the rigor that must exist in the process conceiving an investment plan. Since there is no need to specify the physical space of IXs, the information concerning DE organization in IXs is not considered, until the elaboration of the final Enterprise Structure, and as such, is not presented in the diagram. The Interaction Spaces Diagram of the elaboration of the Enterprise Structure for the “Grow Project Community” BP is presented in Figure 43.

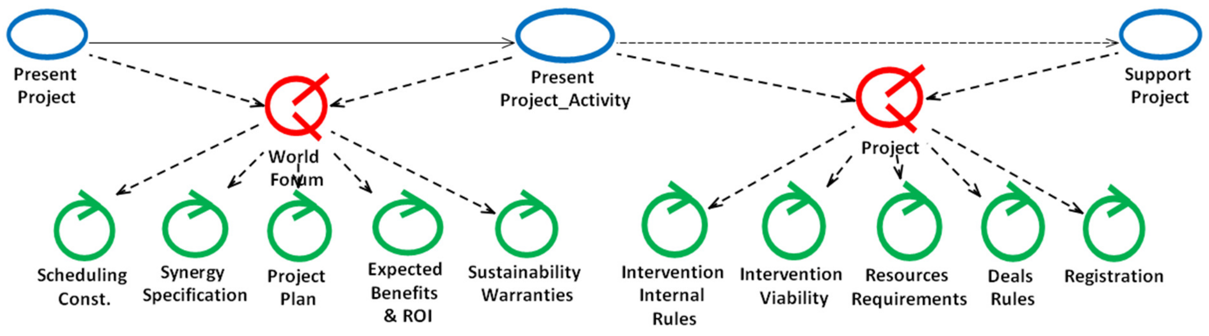


Figure 43. Enterprise Structure Interaction Spaces Diagram.

The “Project” and “World Forum” involve the creation of the project, and also of the customer supportive interaction, to which important *regulations* apply. These are, for the “World Forum” IX: “Scheduling Constraints”, “Synergy Specification”, “Project Plan”, Expected Benefits & ROI”, and “Sustainability Warranties”. These BRs reflect the focus on the financial rigor that the project must comply with. For the “Project” IX: “Intervention Internal Rules”, “Intervention Viability”, “Resources Requirements”, “Deals Rules”, and “Registration”. These BRs reflect the rules that must apply to the predicted types of support, namely concerning deals, interventions, and the *resources* that may be involved.

Data Entities Identification

The DEs which were already identified in Step 1 - Service Design, are now complemented with the “Resources” and “Deals” DEs. The BRs were related to DEs by means of the semantical meaning of each BR, and the more basic DE concepts.

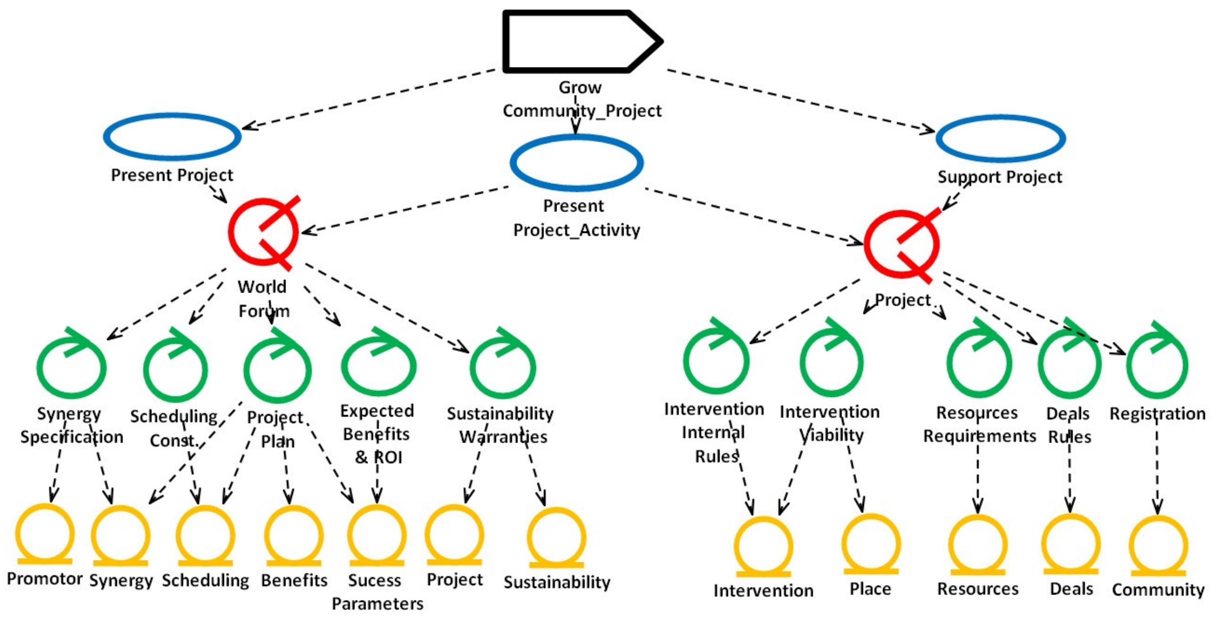


Figure 44. Step 3 – Enterprise Structure of "Grow Community Project" BP.

Figure 44 presents the final Enterprise Structure of the "Grow Project Community" BP, where it is possible to identify the IXs and BRs of the Interaction Spaces Diagram are now related to the DEs that apply in each case. This structure represents the business up to the moment that it was elaborated according to the diagrams produced in order to attain the objective of the project, which was the elaboration of the website User Interface design. This model is supposed to be updated along the continuation of the modeling of the enterprise.

4.4.4. Step 4 - Task Modeling

From the "Grow Project Community" BP we focus on the "Present Project Activity" UT performed by the "Mission Manager", which objective is to publish the distinct forms of support that may be available in each project. The needed information is introduced by the "Project Manager" and "Project Builder" Actors in the "Present Investment Possibilities" (of the "Conceive Investment Project" BP), and "Present Project" UTs (of the "Grow Community Project" BP); and by the "Mission Manager" in the "Plan Intervention" UT (of the "Support Investment" BP). The participation actions were found to be the "Investment", "In-Loco Interventions" and "Deals & Donations", to which the "Mission Manager" could also add more project information. The Task Model, which is the first of the two-step decomposition of the UT in User Intention (first step) and User Interactions (second step) according to the CTT method, was elaborated for two UTs, in order to specify the logic of cooperation concerning the usage of the same *business concepts*, promoting the continuous reuse of components.

Task Model

The Task Model includes the User Intentions of the "Present Project Activity" (left side of Figure 45) and the "Support Project" UTs, namely "Publish Project Information", "Publish Investment Possibilities", "Publish In-Loco Intervention Information" and "Publish Donation & Deals Possibilities", which have a correspondence in the User Intentions of the "Supporter" Actor (right side of the Figure), of "Analyze Investment Possibilities", "Analyze In-Loco Intervention Information" and "Analyze Donation & Deals Possibilities".

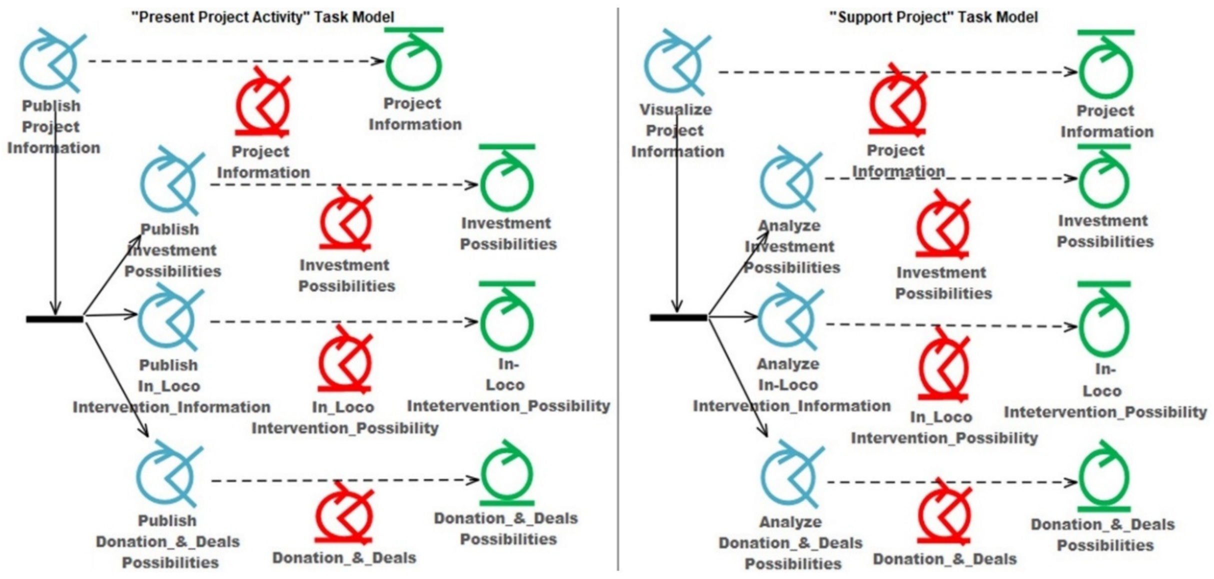

"Present Project Activity" Task Model
Publish Project Information
Project Information
Project Information
Publish Investment Possibilities
Investment Possibilities
Investment Possibilities
Publish In_Loco Intervention_Information
In_Loco Intervention_Possibility
In-Loco Intetervention_Possibility
Publish Donation_&_Deals Possibilities
Donation_&_Deals
Donation_&_Deals Possibilities
"Support Project" Task Model
Visualize Project Information
Project Information
Project Information
Analyze Investment Possibilities
Investment Possibilities
Investment Possibilities
Analyze In-Loco Intervention_Information
In_Loco Intervention_Possibility
In-Loco Intetervention_Possibility
Analyze Donation_&_Deals Possibilities
Donation_&_Deals
Donation_&_Deals Possibilities

Figure 45. Step 4 – Task Model | "Attract … Community"/"Support Project".

The Task Model is presented in Figure 45. The decomposition of the User Intention by affinity in nouns (*business concepts*) and verbs, allows the identification of the "Investment Possibilities", "In-Loco Intervention Information", and "Donation & Deals Possibilities" as new *business concepts*, which are to be included in the "Investment", "In-Loco Intervention" and "Donations & Deals" DEs, respectively. Thus, from these *business concepts,* the attributes of "Publish" and "Analyze" were considered, establishing a direct relation between the User Intentions of the distinct UTs (e.g. "Publish Investment Possibilities" and "Analyze Investment Possibilities"), and promote the derivation and reuse of the "Investment Possibilities", the "In-Loco Intervention Possibility" and "Donations & Deals" Interaction Components (IC).

4.4.5. Step 5 – User Interface Design

The User Interface design targeting the identification of Fields and arrangement of Interaction Components (IC) depends on the elaboration of the Interaction Model, which applies a structure of decision to each User Intention. According to the interaction pattern chosen to specify the elaboration of the Software Architecture of the Information System, the specification allows the collection of all the *data*, and their presentation in Interaction Objects (IO) by means of the support provided by User Interface System Responsibilities (UISR), followed by an operation as demanded by the user, supported by a Database System Responsibilities (DBSR).

Interaction Model

In the example presented in Figure 46, the "Analyze In-Loco Intervention Information" User Intention of the "Support Project" UT (of the "Grow Community Project" BP), is decomposed into a logic of interaction based on the BDD method. The User Story is presented in UML notes and related to the User Interactions, derived from the BDD pseudo-code as presented in Table 7, until reaching the "Type Registration Observations" and "Click Register for Intervention", along with the identified IOs and SRs.

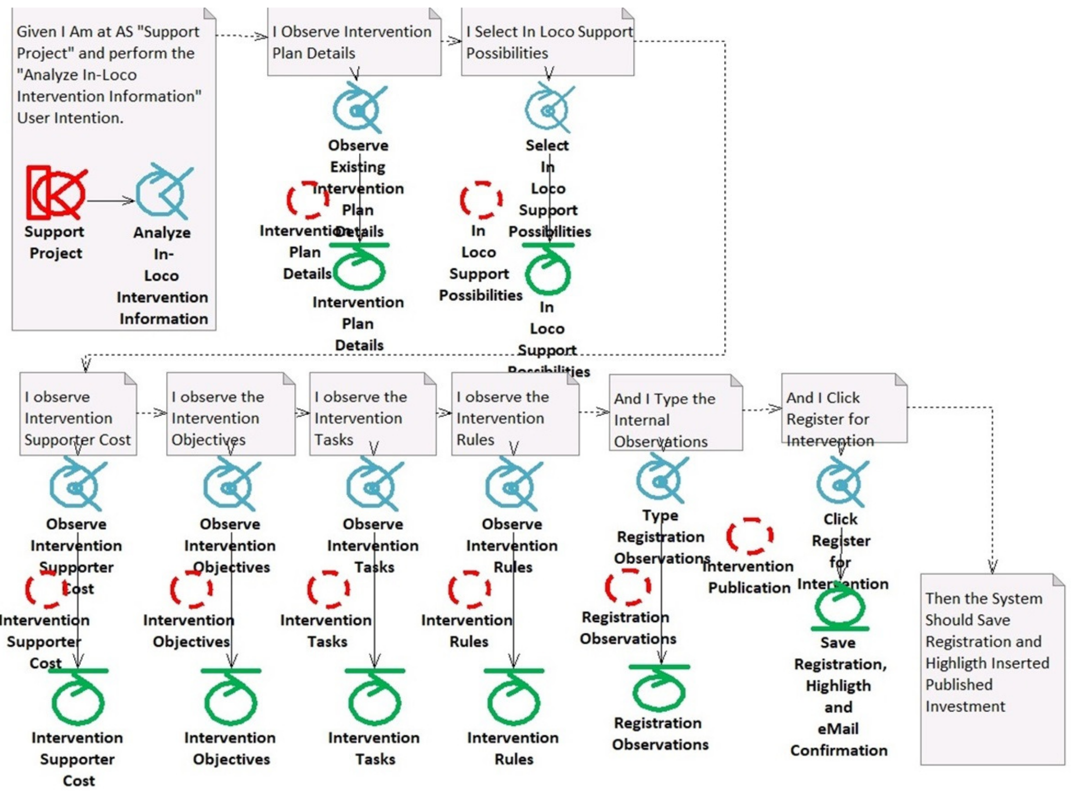

Given I Am at AS "Support Project" and perform the "Analyze In-Loco Intervention Information" User Intention.
Support Project
Analyze In-Loco Intervention Information
I Observe Intervention Plan Details
Observe Existing Intervention Plan Details
Intervention Plan Details
Intervention Plan Details
I Select In Loco Support Possibilities
Select In Loco Support Possibilities
In Loco Support Possibilities
In Loco Support Possibilities
I observe Intervention Supporter Cost
Observe Intervention Supporter Cost
Intervention Supporter Cost
Intervention Supporter Cost
I observe the Intervention Objectives
Observe Intervention Objectives
Intervention Objectives
Intervention Objectives
I observe the Intervention Tasks
Observe Intervention Tasks
Intervention Tasks
Intervention Tasks
I observe the Intervention Rules
Observe Intervention Rules
Intervention Rules
Intervention Rules
And I Type the Internal Observations
Type Registration Observations
Registration Observations
Registration Observations
Intervention Publication
And I Click Register for Intervention
Click Register for Intervention
Save Registration, Highligth and eMail Confirmation
Then the System Should Save Registration and Highligth Inserted Published Investment

Figure 46. Step 5 – Interaction Model | "Analyze … Intervention Information".

User Interface Design

The User Interface reflects the structure of *business concepts* which are involved in the UT, as presented in the Service Design (Section 4.4.1. Step 1 - Service Design). Applied to the "Star Project" this logical structure is the relation of the "Synergy" DE with the remaining *business concepts*. The "Synergy" drives the project including multiple investment channels, which are composed of distinct investment possibilities, interventions to help support the project, and the possibility of donating and establishing deals by supporters who may want to work, provide machinery or any other valuable physical *resources* that they may want to dispense.

The User Interface is elaborated from top-left to right-down following the western calligraphy. In the case that the method is to be used in a social environment where the calligraphy may be e.g. from top-right to left-down, the adaptation of this logic should be considered. The resulting User Interface, including the ICs derived from the remaining User Intentions of the "Support Project" UT, are presented in Figure 47, where the "In-Loco Interventions" IC is highlighted with its concept symbol.

Following the Interaction Model, the "Supporter" Actor may type observations and register for an intervention. The "In-Loco Interventions" IC, the "Registration Observations" and the "Intervention Registration" IOs are highlighted in the Figure using UML stereotype nearby the object as designed using the Canonical Abstract Prototypes (CAP). The presented User Interface structure is the result of the elaboration of the "Present Project" UT Task Model of the "Project Manager" Actor, from which the following ICs were reused: "Project Character"; "Project Sustainability"; "Success Parameters"; "Investment Possibilities"; "In-Loco Interventions"; and "Donations & Deals", according to the ones which were identified in the Task Model of the "Support Project" UT of the "Supporter" Actor.

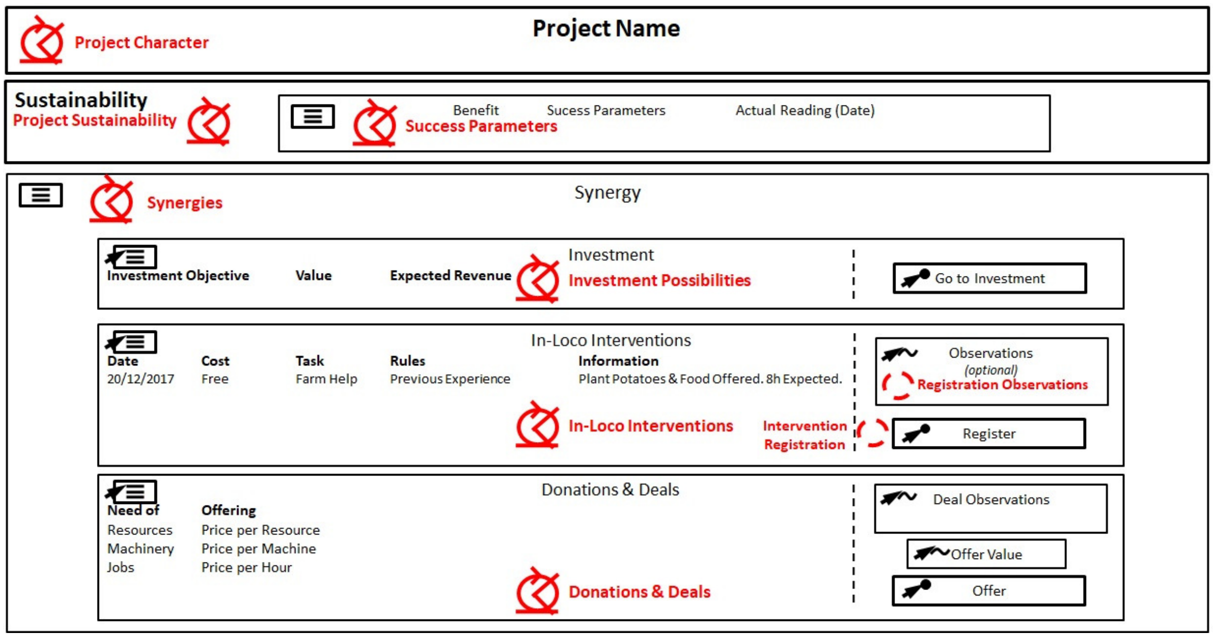


Figure 47. User Interface Example.

The "In-Loco Interventions" IC which has a correspondence with the "Present Project Activity" UT of the "Mission Manager" Actor which is presented in Figure 48. Both User Interfaces reflect the type of reuse which is facilitated by means of the application of the GOALS method, since the "Intervention Public Observations" IO and the "Intervention Publication" IO are what changes in the "In-Loco Interventions" IC, allowing the "Mission Manager" to perform an interaction in the "Registration Observations" IO, in the same way of the interaction in the "Intervention Registration" IO performed by the "Supporter". The remaining components of the IC are maintained, and the structure of ICs of the AS of both *actors* is the same, as long as the components are reused to the User Intentions and User Interactions that they apply.

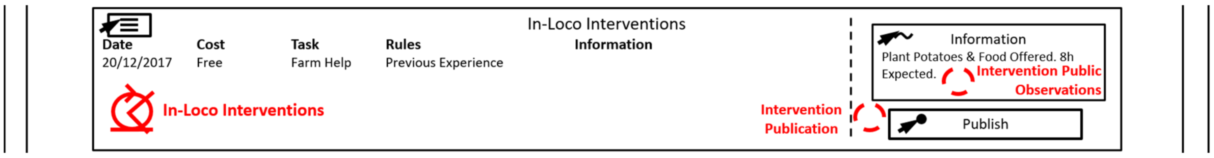


Figure 48. User Interface Reuse Example.

Website Design

The design of the website is oriented by the "Support Project" and "Investment Possibilities Feedback" UTs of the "Supporter" Actor, which have a structure of projects which needs to be browsed so that the user may choose which project to support, and how. Thus, the "Investment Possibilities Feedback" UT refers to the "Investment Possibilities" IC of "Support Project" UT presenting a project still not approved, so that the user may "Introduce Textual Feedback", promise "Participation" or "Donations & Deals", compromising on how much money he would like to invest. Figure 49 presents the resulting website front-end design which allows browsing the places and projects. A new IC "Project Design" was added in order to provide a visual impact of the project, namely in terms of the service provided to the community. In the example, the GOALS notation was used to exemplify the objective of the IC, to which are added the already known supporter interaction possibilities, now in terms of a pre-investment poll.

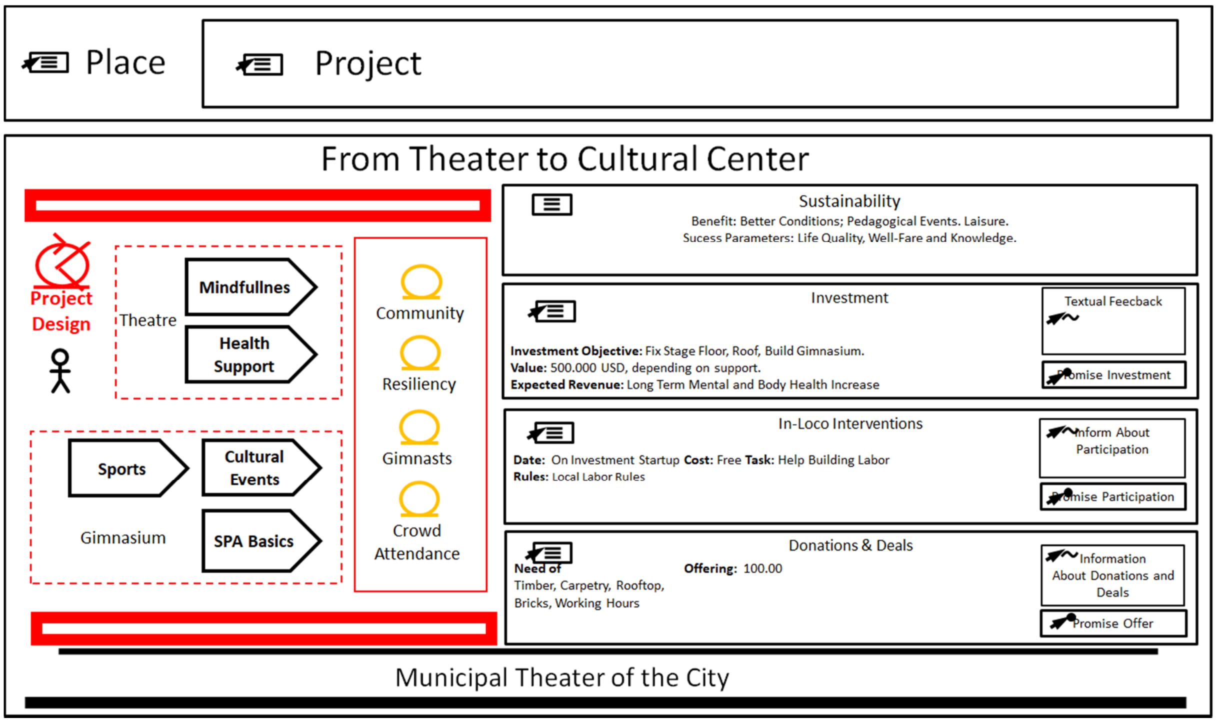

Place
Project
From Theater to Cultural Center
Project Design
Theatre
Mindfullnes
Health Support
Sports
Cultural Events
Gimnasium
SPA Basics
Community
Resiliency
Gimnasts
Crowd Attendance
Sustainability
Benefit: Better Conditions; Pedagogical Events. Laisure.
Sucess Parameters: Life Quality, Well-Fare and Knowledge.
Investment
Investment Objective: Fix Stage Floor, Roof, Build Gimnasium.
Value: 500.000 USD, depending on support.
Expected Revenue: Long Term Mental and Body Health Increase
Textual Feecback
Promise Investment
In-Loco Interventions
Date: On Investment Startup Cost: Free Task: Help Building Labor
Rules: Local Labor Rules
Inform About Participation
Promise Participation
Donations & Deals
Need of Timber, Carpetry, Rooftop, Bricks, Working Hours
Offering: 100.00
Information About Donations and Deals
Promise Offer
Municipal Theater of the City

Figure 49. Step 5 – Website Design of the “Star Project”.

4.4.6. Step 6 – Business Logic Structuring

The Business Logic structuring focus on the “In-Loco Intervention Possibility” User Intention, which provides support structure to the “Attract Project Community” and “Support Project” UTs, distinguished in terms of functioning according to the User Stories of the Interaction Model of each UT, which function as *actors* “permissions” over the management of each *business concept*. The structuring is carried out according to the affinity logic of the *business concepts*, promoting the reuse of existing DEs.

Structuring by Affinity of Business Concepts

The Business Logic components that support Interaction Components (IC), usually manage more than one field and eventually more than one DE. The components that support Interaction Objects (IO), usually manage one Field if they are User Interface System Responsibilities (UISR), and more than one DE if they are Database System Responsibilities (DBSR). The relation of the User Interface components with the Business Logic is already established, providing meaning to each of the components of the Business Logic, which are now to be related to the existing DEs, according to the perception that exists of the managed *business concepts*, independently of the reading and writing implementation that may be used to implement each Business Logic component.

The structure present in Figure 50 contains the “In-Loco Intervention Possibility” UISR, which serves as an interface to the remaining UISRs and DBSRs derived by the “Analyze In-Loco Intervention Information” and “Publish In-Loco Intervention Information” Interaction Models. The UISRs are responsible for collecting the information which is presented in the User Interface. The “Save Intervention, Highlight Confirmation” and “Save Registration, Highlight and eMail Confirmation” BDSRs are responsible for saving the *data* in the Database and produce feedback to the User Interface.

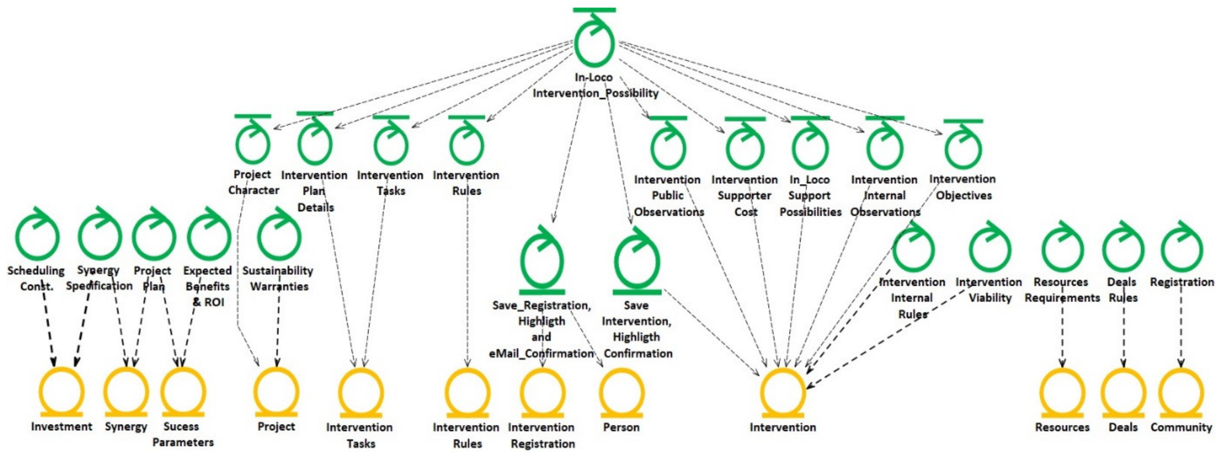

In-Loco Intervention_Possibility
Project Character
Intervention Plan Details
Intervention Tasks
Intervention Rules
Intervention Public Observations
Intervention Supporter Cost
In_Loco Support Possibilities
Intervention Internal Observations
Intervention Objectives
Scheduling Const.
Synergy Specification
Project Plan
Expected Benefits & ROI
Sustainability Warranties
Save_Registration, Highligth and eMail_Confirmation
Save Intervention, Highligth Confirmation
Intervention Internal Rules
Intervention Viability
Resources Requirements
Deals Rules
Registration
Investment
Synergy
Sucess Parameters
Project
Intervention Tasks
Intervention Rules
Intervention Registration
Person
Intervention
Resources
Deals
Community

Figure 50. Step 6 – Business Logic Structuring | "… Intervention Possibility".

Each component is related by means of affinity of the Fields of the DEs where these Fields belong. In the case of the DBSRs, the relation is established by means of the prediction of what are the DEs which it algorithm will need. Furthermore, the Enterprise Structure "Sustainability Warranties", "Intervention Viability", and "Intervention Internal Rules" BRs are also included. The "In-Loco Intervention Possibility" UISR is related to those BRs by means of the "Attract Project Community" and "Support Project" UTs IXs, namely "Project" and "World Forum".

4.4.7. Step 7 – Database Structuring

The Database structuring is the detail of the domain model and the Data Entities (DE) which it contains in terms of Fields and new DEs, following their derivation in the previous Steps. The Fields are placed in the DEs by means of the affinity of their meaning. These Fields are class attributes, which may contain *data*, or may be calculated. In the cases when they are calculated, they may represent states of the system which characterize the DE. These states are independent from the states of the system that highlight the result of User Interactions, although these states may depend on them in function of the defined User Stories, and therefore, the related Business Logic components (UISR, DBSR and BR).

Structuring by Affinity of Business Concepts

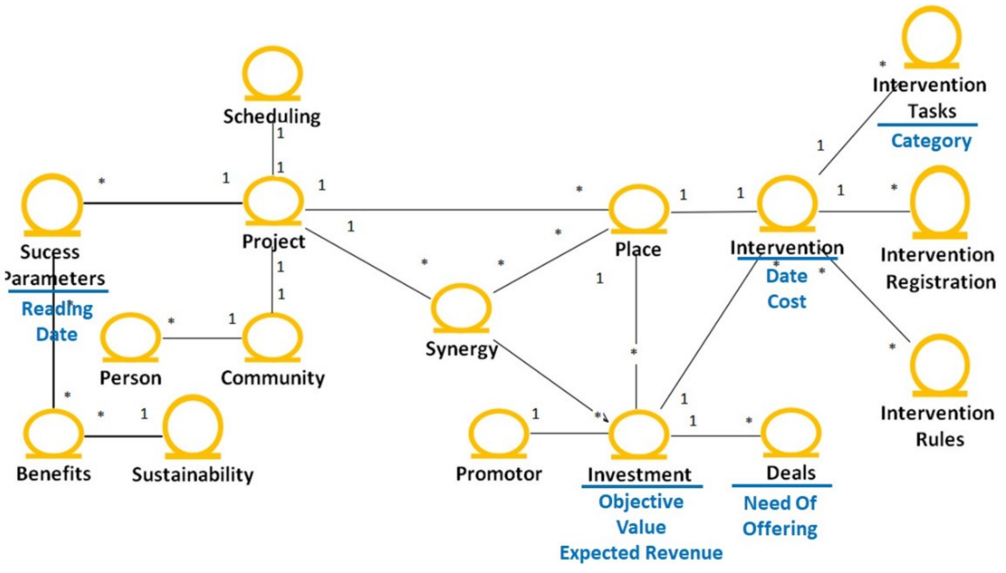


Figure 51. Step 7 – Database Structuring.

The example in Figure 51 presents the Database structure after the elaboration of Step 6 – Business Logic Structuring, to which the DEs “Intervention Tasks”, “Intervention Registration”, and “Intervention Rules” where added as a result of Step 5 – User Interface Design, along with new Fields, namely: “Reading” and “Date” of the “Success Parameters” DE; “Objective”, “Value” and “Expected Revenue” of the “Investment” DE; “Date” and “Cost” of the “Intervention” DE, and “Information” of the “Intervention Tasks” DE.

4.4.8. Step 8 – Software Architecting

The Enterprise Structure elaboration is the arrangement of the diagram using the relations which were already previously established. Usually the relation between DEs is not represented, since at this stage, their positioning by means of affinity is sufficient in order to understand the functioning of the system. By means of arranging the diagram by affinity, the software modeler will usually be able to find the more appropriated spaces to position each Software Architecture component while maintaining the readability of the diagram, which will usually fit an A3 sheet.

Figure 52 presents the Software Architecture of the “Investment Feedback” and “Support Project” Aggregation Spaces (AS), which support the UTs with the same name of the “Grow Project Community” BP, which is the specification of front-office of the website. The “Project Information” IC is the common support structure of both UTs. Each IC has a UISR which serves as an electronic interface to the UISRs and DBSRs that support the User Interactions. The Business Rules (BR) of the Enterprise Structure which refer to the Interaction Spaces (IX) where the supported UT occurs, “Project” and “World Forum” are also presented in the architecture.

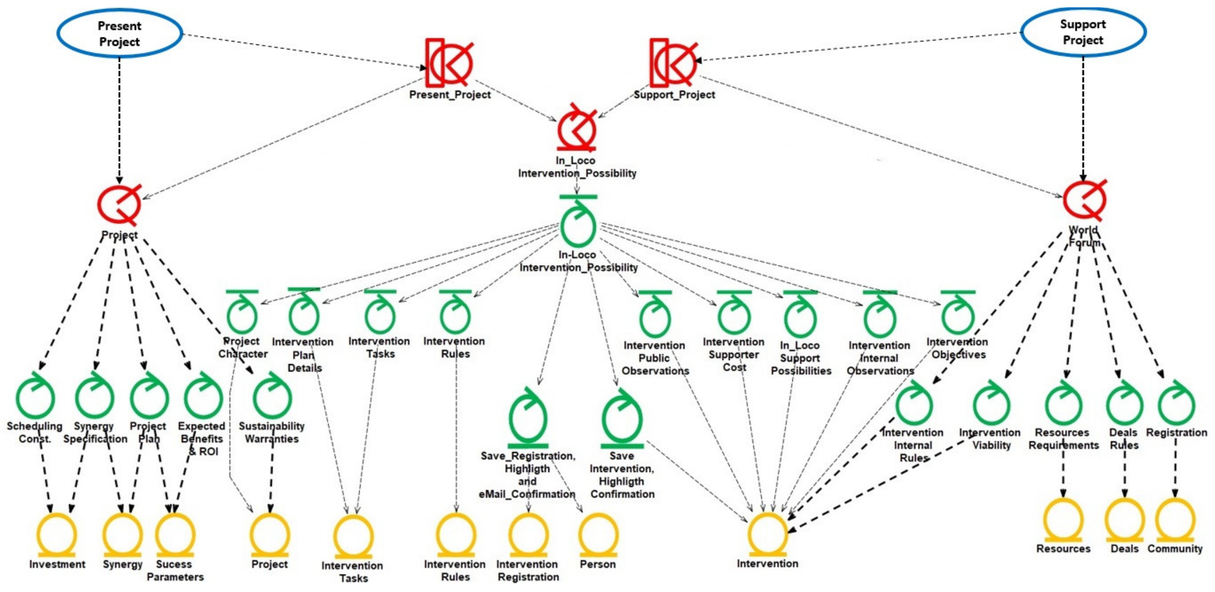


Figure 52. Step 8 – Software Architecture.

Notice that the “Star Project” complexity is elevated when compared e.g. to the “iKiosk” project presented in Chapter 4. The Goals Process, Method and Techniques. That happens because of the number of *business concepts*. For this reason, the number of components is high, yet, nevertheless, all the Fields of the User Interface are represented enabling traceability and also facilitating implementation control. The Software Architecture may be used in order to assign and create a collaboration concerning each component development within the software development team.

4.4.9. Case Study Conclusions

The presented case study reflects how the method may be used in order to specify the organization of enterprises including envisioning the facilities and buildings that they may occupy. The case study allows the understanding which business parts (or logical facilities) of the enterprise are supported by automated software and which parts are not, observed in the holistic perspective of the *Business Owner*, the *Service Designer* and the *Business Process Manager* during the Analysis phase.

The development moments of the "Star Project" are presented in Table 9. There were 2 formal meetings with the *Business Owners*, which also performed the roles of *Service Designer* in two meetings (08-02-2017 and 23-02-2017). The project followed the Method Steps and had demanding phases concerning the final steps of each Analysis and Design phases, in a total of 86 working hours including the client time.

Table 9. "Star Project" Development Process.

#	Date	Stakeholders	Phase	Steps	H
1	27-09-2016	*Business Owner* *Software Architect*	Analysis	*User Interface Drafts of Business Owner* 1. Service Design	4
2	20-10-2016	*Software Architect*	Analysis	1. Service Design 2. BP Design	10
3	08-02-2017	*Business Owner* *Service Designer* *Software Architect*	Analysis	2. BP Design	3
4	12-02-2017	*Software Architect*	Analysis	2. BP Design 3. Enterprise Structure 3. Enterprise Structure 4. Task Model	20
5	23-02-2017	*Business Owner* *Service Designer* *Software Architect* *User Interface Designer*	Analysis & Design	1. Service Design 2. BP Design 3. Enterprise Structure 4. Task Model 5. User Interface	8
6	18-03-2016	*Software Architect* *User Interface Designer*	Design	4. Task Model 5. User Interface 6. Business Logic 7. Database 8. Software Architecture	40
7	14-04-2017	*Business Owner* *Software Architect*	Analysis & Design	5. User Interface *Project Validation*	1
					86

The number of hours spent in the elaboration of the Design phase in moment 6, 40, reflects the complexity of the project, which is augmented by the fact that the conception of the enterprise was completely new, without a reference model of any other enterprise besides what the websites of capturing organizations present. The presence of a User Interface Designer revealed to be important in order to help managing the complexity generated from the derived components and their design and structuring. Independently of the managed complexity, the Method Steps were followed in a standard way, and the client approved the final User Interface design, which was the main objective of the project.

4.5. Conclusions

This chapter presented the process that applies a single method and language, including the bridging and structuring techniques which elaborate the Software Architecture based on the Business Architecture (the Enterprise Structure). Two case studies were presented, which complement each other by first presenting the conception of each step and its elaboration, and after, highlighting the more important aspects of each step and their related decisions, in the process of elaborating the business and software models.

The application of the method allowed the design of the *service* and *business processes* of the enterprise, including the used *spaces*, applied *regulations* and used *data*. From this information, the business is structured, and based on this structure, human-centric techniques are applied, structuring of the User Interface of the Information System, its Business Logic, which is complemented with a Domain Model from which the Database is derived, achieving the full specification of the Software Architecture for the BPI. Whether a single BP-sized improvement, or all the BPs of the enterprise.

The final Software Architecture reflects the business *requirements* which are being supported, since the relation from the User Interface (an Aggregation Space (AS)) to the User Task (UT) is of one-to-one (by default), and each UT decomposition in terms of intentions and interactions, and their relation to the structure of the Information System is also of one-to-one. The remaining business *requirements* are expressed in the Software Architecture by means of Business Rules (BR) in the Business Logic, and by Data Entities (DE) in the Database, which processing is ensured by means of the Interaction Space (IX) which is used (architecturally) “inherited” by the Aggregation Space, as a reinforcement of the (one-to-one) traceability of (business) UT and (software) AS.

The resulting Business and Software Architectures may be used to organize the business and its people, and also the implementation of the software that will support that organization. The Software Architecture may be used to define implementation priority, and also affection in function of each team member software development competences. The process may be applied for smaller or larger software development teams, and smaller or larger BPIs, knowing in advance, that smaller projects have increased probabilities of success [The Standish Group, 2013]. For this reason, the EDP should allow software project size to be evolved adequately, according to the development team capability, privileging effectiveness in first place, and efficiency in second, as efficiency depends also on experience, a reason to favor preliminary positive results (slower projects), as these may induce efficiency improvement (faster projects) in the long-term.

5. Goals EA Template

The usage of architectural applications (or tools) for the elaboration of the models of a software development project gives the modeler advantages in terms of model organization. Concerning the elaboration of GOALS diagrams, an architectural tool allows the creation of a template that includes classes (GOALS concepts), their stereotypes (symbols), and the relation between concepts, which are simplified to dependency and association.

The main difference from an architectural tool to a design tool, is the maintenance of a single representation of each class along all the diagrams where those classes may be instantiated in terms of its structure and also its associations. This eliminates the need to update the design of the diagram when a class or a relation changes or is deleted, as that change will also happen in the remaining diagrams automatically. Thus, an architectural application allows the elaboration of multiple diagrams where the same classes are used, yet only presenting the ones which are needed at each moment for the conception of the system. Moreover, each class may also include diagrams which represent their internal structure, a drill-down that allows an enhanced organization of the diagrams of the project.

The tool elected to model GOALS is Enterprise Architect v13.5 Ultimate Edition [Sparx Systems, 2017], which provides support for the application of the UML 2.1 language, providing a set of features which allow the organization of all the information of a project, namely the stereotypes and diagrams, which we present as follows, focusing the application of GOALS.

5.1. Organization

The Enterprise Architect (EA) basic features are the “Tool Box”, which provides the classes and relationships which may be used to elaborate the diagrams, the “Project Browser”, which allows the identification of the created diagrams and components, the “Diagram” zone in which the modeler elaborates each diagram, and the “Tool Bar” which provides an extensive set of tools concerning overall EA functionality, separated in tabulators (“Start”, “Design”, etc.).

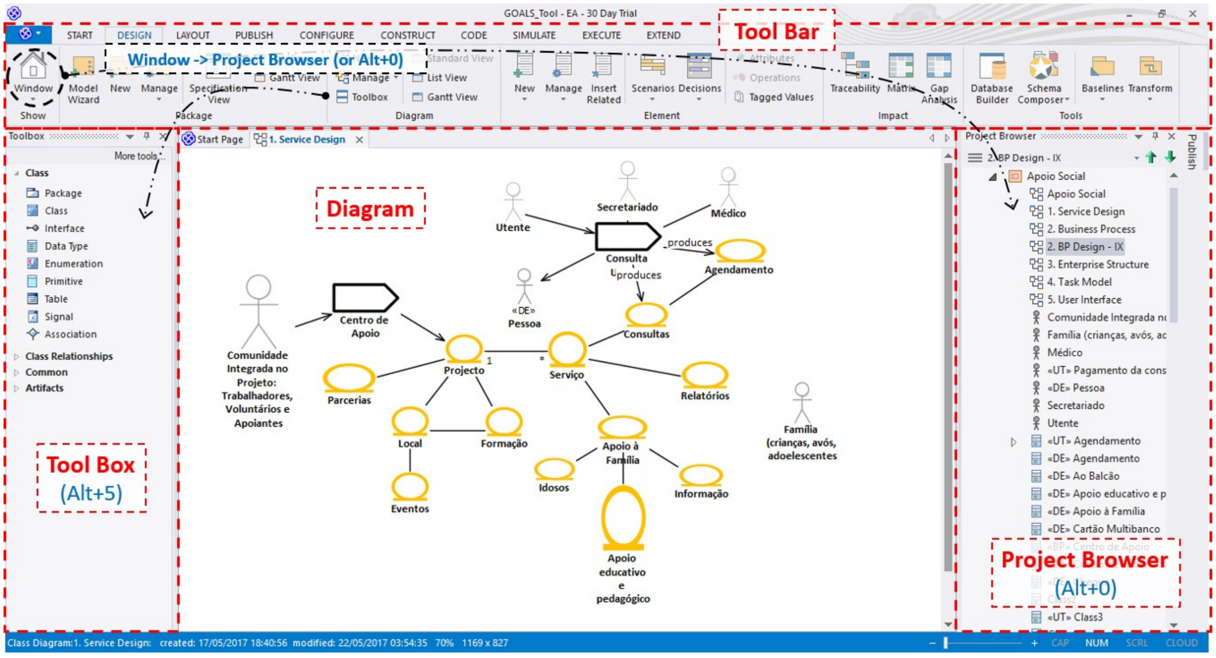
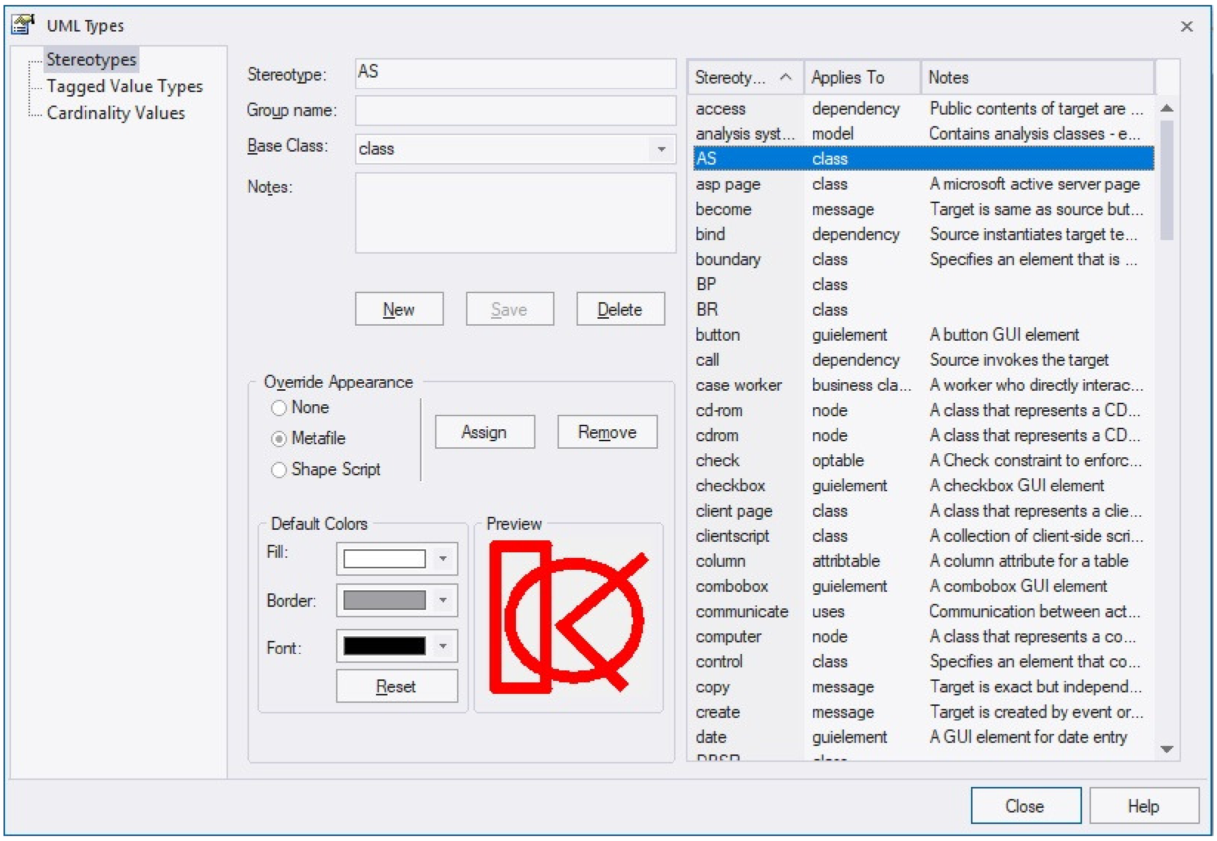


Figure 53. Enterprise Architect (EA) Organization.

5.1.1. Stereotype Organization

Each class type should have its own stereotype in order to facilitate its identification in the diagrams. Each GOALS concept is a UML class type to which a stereotype may be defined using the "UML Types" tool, available from the "Tool Bar", at "Configure -> UML Types", as presented in Figure 54. In the "UML Types" tool, the modeler may create a new "Stereotype", typing its designation and "Base Class" (which should the generic "class"), and then clicking the button "Save".

In order to specify a file with the image of the stereotype, the modeler should edit the stereotype by selecting it from the existing "Stereotypes" (e.g. the "AS" stereotype as shown in the Figure), and select its metafile by specifying its location in the local hard drive. The stereotype file should be of the type ".EMF" (Enhanced Metadata File) or ".WMF" (Windows Meta File). We suggest that the image has transparency so that only the stereotype lines are visible in the diagram, and not the background (usually a white rectangle), in order to improve readability.



Stereotype: AS
Group name:
Base Class: class
Notes:

Override Appearance: ○ None [✓ Metafile] ○ Shape Script

Fill:
Border:
Font:

Stereoty...	Applies To	Notes
access	dependency	Public contents of target are ...
analysis syst...	model	Contains analysis classes - e...
AS	class	
asp page	class	A microsoft active server page
become	message	Target is same as source but...
bind	dependency	Source instantiates target te...
boundary	class	Specifies an element that is ...
BP	class	
BR	class	
button	guielement	A button GUI element
call	dependency	Source invokes the target
case worker	business cla...	A worker who directly interac...
cd-rom	node	A class that represents a CD...
cdrom	node	A class that represents a CD...
check	optable	A Check constraint to enforc...
checkbox	guielement	A checkbox GUI element
client page	class	A class that represents a clie...
clientscript	class	A collection of client-side scri...
column	attribtable	A column attribute for a table
combobox	guielement	A combobox GUI element
communicate	uses	Communication between act...
computer	node	A class that represents a co...
control	class	Specifies an element that co...
copy	message	Target is exact but independ...
create	message	Target is created by event or...
date	guielement	A GUI element for date entry

Figure 54. EA - Stereotypes Organization Tool.

For the creation of the .EMF or .WMF file, we suggest using the Microsoft PowerPoint®, by: drawing the image; selecting the image; selecting the option "Save as Picture…" from the drop-down menu when right-clicking (for right handed people) the image; and then choosing the image format, in order to experiment it in EA. We suggest the creation of the stereotypes for the 12 GOALS concepts (using the in-brackets acronyms): Business Process (BP), User Task (UT); Interaction Space (IX); Business Rule (BR); Data Entity (DE); User Intention (UserIntention); User Interaction (UserInteraction); Aggregation Space (AS); Interaction Component (IC); Interaction Object (IO); User Interface System Responsibility (UISR); and Database System Responsibility (DBSR).

5.2. Diagrams

In EA, diagrams are managed in the “Project Browser”, using a hierarchical structure of views that contain the diagrams of the project. We suggest the creation of a view with the name of the project, in which all the diagrams will be contained.

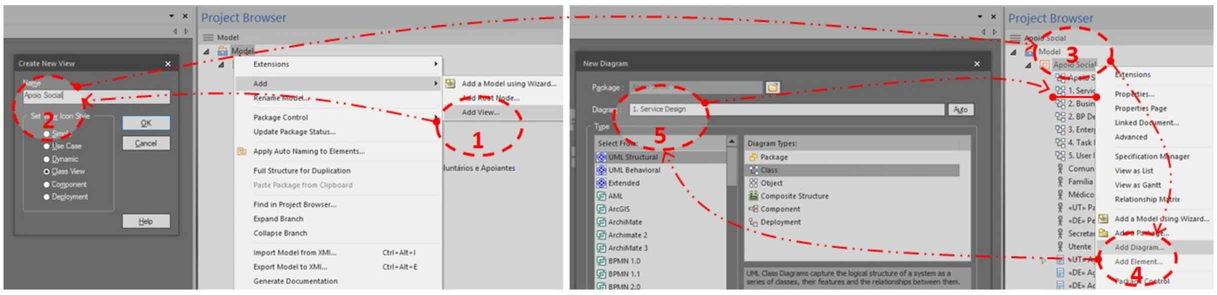


Figure 55. EA - Package and Diagram Creation.

Figure 55 exemplifies the situation, where the first step (highlighted as “1”) is selecting the option of “Add View”, and naming the view (“2”), in this case “Apoio Social” (the name of the project). After the view is created, diagrams may be added by selecting “Add Diagram…” (“4”), and naming the diagram, in this case “1. Service Design” (“5”), which will after appear under the view “Apoio Social”, as highlighted with an horizontal bulleted line

Diagrams may be created in two ways. By means of using the “Insert Diagram…” option (as presented in Figure 55), or by setting a class as a “Composite Element”, in which case, the diagram is created as a dependency of the class. This situation is presented in Figure 56, where after selection the option “Composite”, a new “Child Diagram” will be created as a dependency of the class (in this case “Agendamento”), with the same name. The diagram will automatically be opened in the “Diagram” zone, empty, in order to be elaborated.

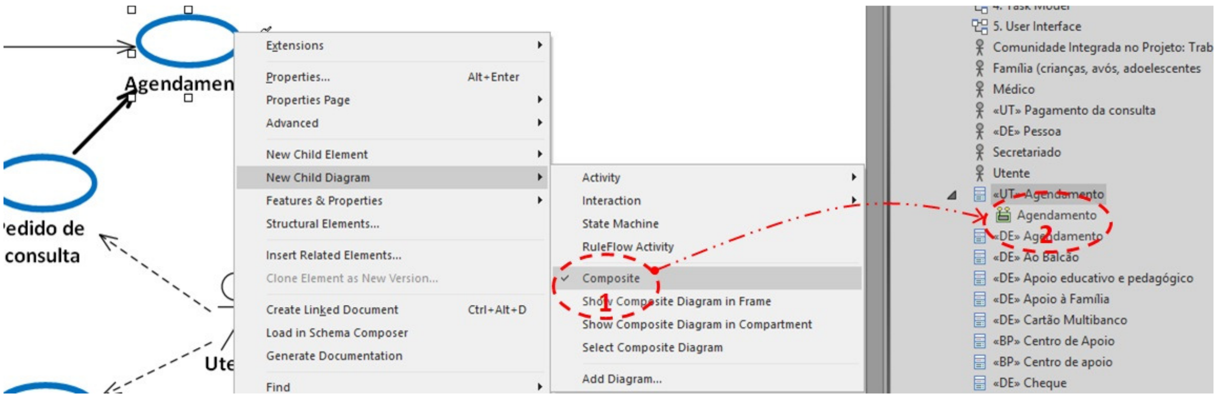


Figure 56. EA - Diagram Creation.

Class Creation

With the diagram open, the modeler may add classes and *actors*, by means dragging them from the “Class” tool from the “Tool Box” to the diagram, to which a name should be given (e.g. “Pessoa”), and the stereotype defined (e.g. “DE”). The modeler may also duplicate classes of the diagram by copying an existing class as a new class, using the “Paste Element(s) as New” tool from the “Edit” drop-down menu (or using the “Ctrl + Shift + V” command after “Crtl + C”), and changing its name. Both class creating dialogs are presented in Figure 57.

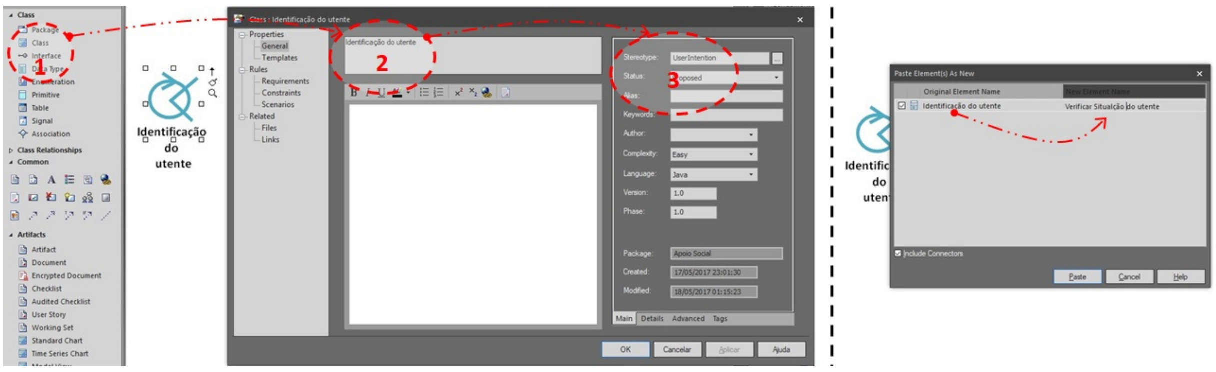


Figure 57. EA - Class Edition and Duplication (only changes the name) Tools.

In the class edition tool, the modeler will also find other structural aspects of the class, namely the attributes management (in the "Detail" separator) which are used to define the Fields of DEs, and the relation of the class with other classes (in the "Link" separator).

Relations between Classes

After two or more classes are already added to the diagram, these may be related between each other. Since GOALS is oriented to the modeling by hand, some adaptations must be made in order to support its modeling using software architectural tool. The relations are of dependency and association:

- The "Dependency" defines that one class depends on the other class existence and good functioning in order to properly work. This is the structural relation that applies between each of the GOALS concepts, independently of business or software.
- The "Association" relates classes in order to compose its meaning, using a cardinality that defines for each of the predicted instantiations of one class, how many may exist from the other class that composes the relation. It applies to Data Entities (DE). It also applies to the relation between User Tasks (UT), for the elaboration of the Step 2. Business Process design. When the relation is conditional, the dependency relation should be used.

Figure 58 presents the 3 types of relation. The dependency (on the left), where the Business Process depends on the User Tasks, which depend on the Interaction Space. The association (in the middle), where Data Entity (DE) class "Projecto" is associated to the DE "Serviço" with a cardinality of 1 to many (represented as 1 and *). The remaining DEs are associated with no cardinality (as an unfinished diagram). The directional (source -> destination) association (on the right) defining the flow of a BP execution, which in the case of the conditional relation (from "Agendamento" to "Ir à Consulta") is represented with the dependency.

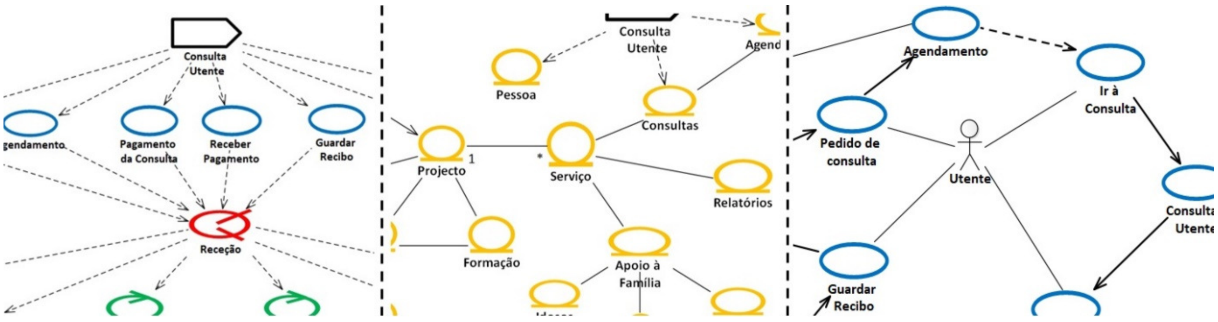


Figure 58. EA - Relations between Classes.

When existing components are added to existing diagrams, the relation which they have with other components are also presented in the diagram by default. However, in some diagrams, it may be necessary to hide some of the relations of the components in order to remove complexity and make it more readable. Two examples are the relations of UTs of the BP design, and the association of DEs in the Enterprise Structure and the Software Architecture. Those relations may be hidden by means of right clicking the relation with the right mouse button (if right handed, if left handed-left button), and setting the Visibility to "Hide Connector", as presented in Figure 59.

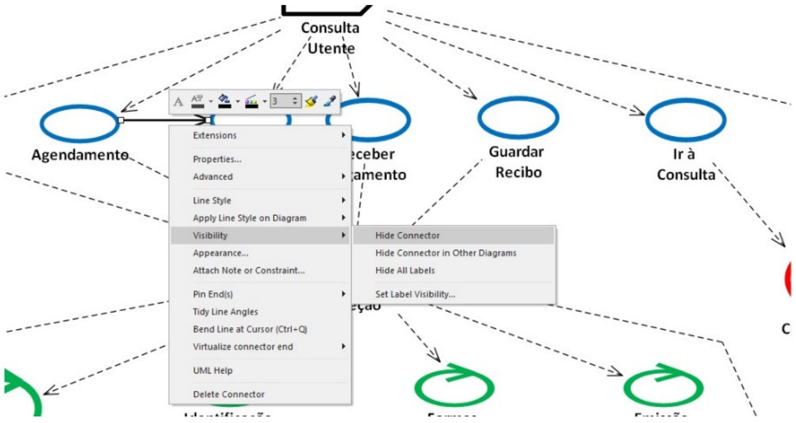


Figure 59. EA - Setting the Visibility of Relations.

5.3. Architectural Views

The possibility of managing classes and all their relations with other classes in each different diagram where it is represented, allow the elaboration of enhanced architectural views, which gather the information from multiple diagrams, in order to provide a holistic perspective of the system being modeled. In GOALS, these diagrams are the Enterprise Structure and the Software Architecture, and are the main reason that justified the usage of an architectural tool for the elaboration of a software development project.

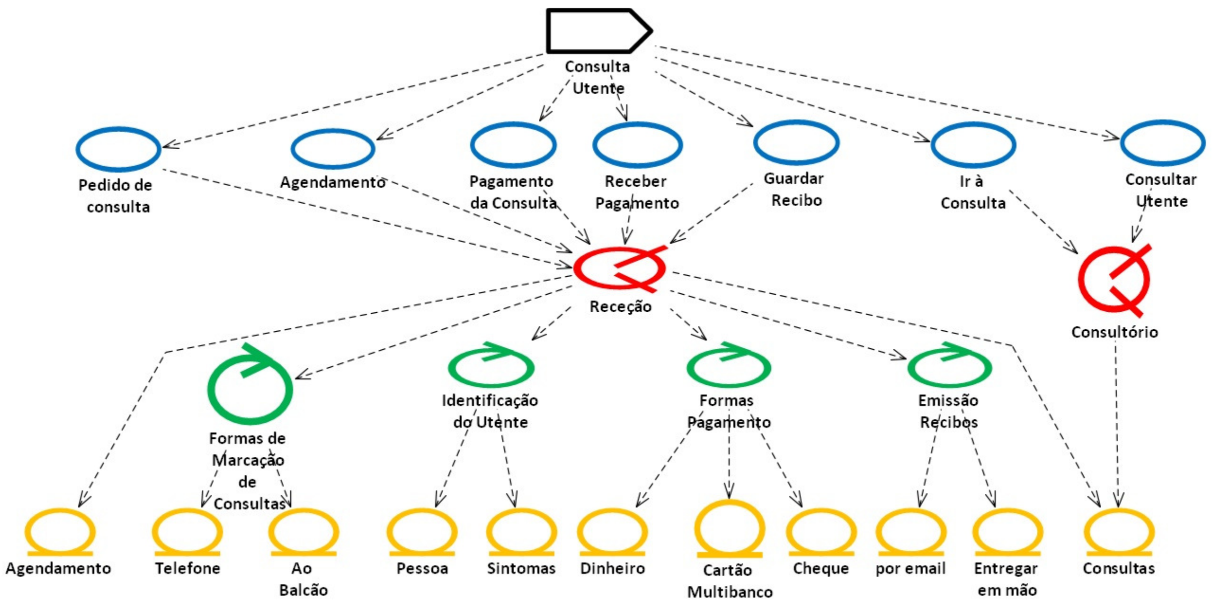


Figure 60. Enterprise Structure as an example of Enhanced Architectural View.

These diagrams are enhanced architectural views that provide a privileged perspective of the system which in the case of GOALS, include all the previously generated components. These diagrams are useful as a base of knowledge that allows the identification namely of, missing relations, duplicated or wrong relations, missing components, and by means of arranging the classes by affinity, also facilitates the identification of lack of organization in classes names.

Figure 60 presents an Enterprise Structure example, which contains 25 classes, and 26 relations of dependency. In this diagram, 15 relations between Data Entities (DE) were hidden in order to facilitate its reading. In the case that the relations do not harm the readability of the diagram, then they may still be presented, which is the case of the 4 flow associations between User Tasks (UT), as they are useful to understand the functional logic without the need to open other diagrams, in this case, the Business Process (BP) design. Thus, this type of diagram summarizes all the previous work, and also serves as a menu to navigate to the internal diagrams of composite elements, namely of the UTs, which in the specific case of GOALS, are from where the design of the system continues.

5.4. Conclusions

This chapter presented how the *Enterprise Architect* (independently of the role that he plays: *Business Owner*, *Service Designer* or *Business Process Manager*, *Software Architect*, *User Interface Designer* and *Software Developer*. may take advantage of the usage of an architectural modeling application in order to develop the 8 models of the system that reflects the business and the software components organization.

6. Validation

The validation of our method is considered in the perspectives of its internal consistency, and of its power of expression. The validation of the GOALS language internal structure cross-consistency targets the validation of our research hypothesis, and also allows the space for the classification of the possible relation types between the language's concepts. Both validation perspectives reflect our contribution concerning the effectiveness, of producing a representative business and software traceable model, and the efficiency of producing a software structure based on a consistent and clear relation between models using a single modeling language.

The cross-consistency validation presented in this chapter is a second and extended version of the validation published in [Valente et al., 2017], in which the validation is extended to the method, and related to the Enterprise Structure (business and hybrid) and Software Architecture (software-specific) concepts, also highlighting the human activity, User Interface, Business Logic, and Database layers and their concept's relations. The power of expression is validated by means of the comparison of the GOALS language with other development methods, in terms of: the assertiveness of the *requirements* elicitation; and comprehensiveness of representation of the business and software in scope and detail. This involves comparison by means of mapping the concepts of the methods analyzed in Chapter 2. State of The Art, with the GOALS concepts, in every case that a comparison applies, and modeling the business, *requirements* or software presented structures using the GOALS language, highlighting the differences. In this comparison, we also include an extended comparison with the DEMO structure which was previously published in [Valente et al., 2016b].

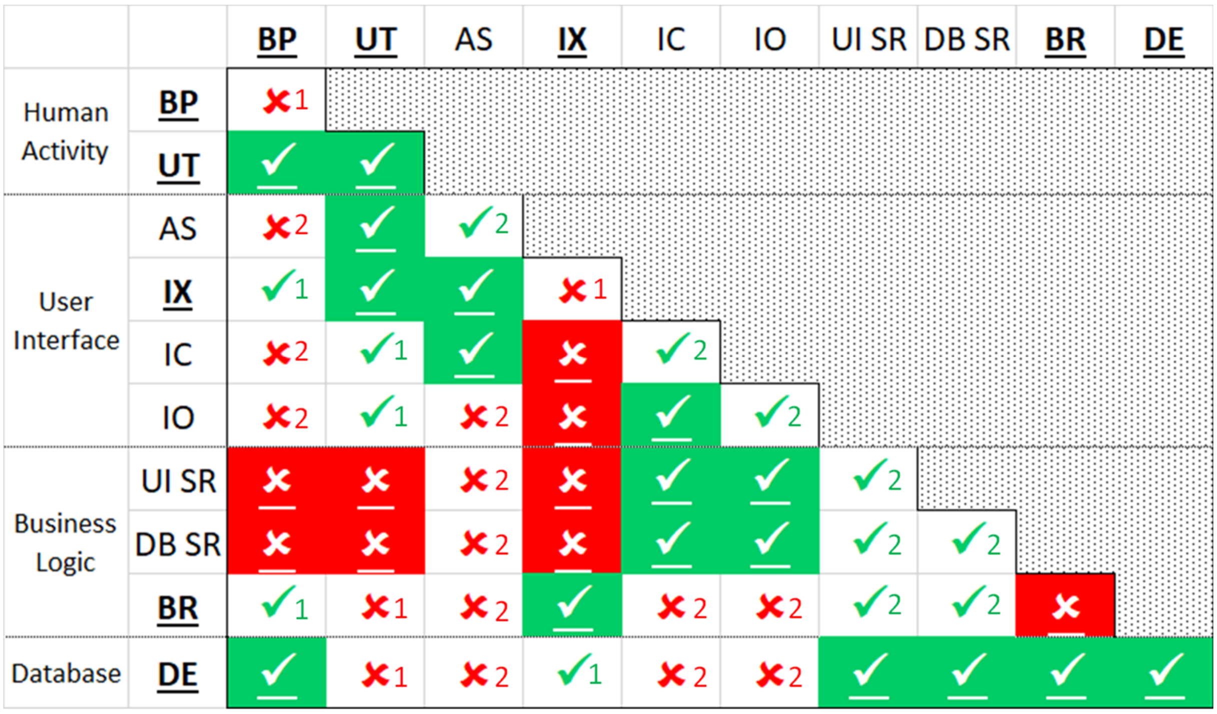
The analysis of the GOALS process in terms of development effort estimation [Alves et al., 2013], previously referenced in Section 1.3. Research Question, is out of the scope of the present validation, which only referrers to the method and language.

6.1. Cross-Consistency

Our research was based on the question of (Q1) "*If it is possible to establish a modeling language that can be applied by a single method which is representative of the organized business activity and its supportive software system?*". And by placing the hypothesis (H1) that it "*Is possible by means of a cross-consistent relation of business and software concepts, and by modeling the business human activity and relating it to software-specific components of the software model by means of the application of architectural patterns*". The validation of H1 is carried out by means of the validation of the internal consistency of the structure of the GOALS language, and by the application of architectural patterns and restrictions (the method) that allow the Information System implementation from a business model, ensuring traceability.

The cross-consistency between concepts is based on the application of the Cross-Consistency Assessment (CCA) [Ritchey, 2015] method and the application of the General Morphological Analysis (GMA) to the relation between GOALS concepts. The objective is to find an "inference model" out of the relations analysis which ensures integrity and clarity, considered as the "Solution Space", from within the remaining contradictory or incompatible relations. In the CCA which was carried out, the GOALS concepts are considered without parameters, as the only relation being considered is the UML *usage* of one concept by the other, and not the compatibility of states of these concepts once instantiated as components, which is a task of the responsibility of the *Software Developer* concerning the programming of the model.

The CCA was applied to the five software-specific components of the Software Architecture: Aggregation Space (AS), Interaction Component (IX), Interaction Object (IO), User Interface System Responsibility (UISR) and Database System Responsibility (DBSR); the two business-specific concepts: Business Process (BP) and User Task (UT); and to the three hybrid Enterprise Structure components: Interaction Space (IX); Business Rule (BR) and Data Entity (DE) . The Cross-Consistency Matrix (CCM), and the resulting types of relations are presented in Figure 61 (Enterprise Structure concepts are underlined), from BP to DE, according to an UML *usage* relation i.e. the component on top uses and depends on the component on the lines below to properly work. The concepts are organized according to the their character: **Human Activity**, which refers to the daily organized actions performed by humans within the business context; **User Interface**, which are the components that present and provide information management tools; **Business Logic**, which refers to the programmable classes; and, **Database**, which refers to the tables and fields and relations between them, of the Information System.



		BP	UT	AS	IX	IC	IO	UI SR	DB SR	BR	DE
Human Activity	BP	✗1									
	UT	🟩 ✓	🟩 ✓								
User Interface	AS	✗2	🟩 ✓	✓2							
	IX	✓1	🟩 ✓	🟩 ✓	✗1						
	IC	✗2	✓1	🟩 ✓	🟥 ✗	✓2					
	IO	✗2	✓1	✗2	🟥 ✗	🟩 ✓	✓2				
Business Logic	UI SR	🟥 ✗	🟥 ✗	✗2	🟥 ✗	🟩 ✓	🟩 ✓	✓2			
	DB SR	🟥 ✗	🟥 ✗	✗2	🟥 ✗	🟩 ✓	🟩 ✓	✓2	✓2		
	BR	✓1	✗1	✗2	🟩 ✓	✗2	✗2	✓2	✓2	🟥 ✗	
Database	DE	🟩 ✓	✗1	✗2	✓1	✗2	✗2	🟩 ✓	🟩 ✓	🟩 ✓	🟩 ✓

Figure 61. Cross-Consistency Matric of the GOALS Concepts.

From the validation of the relation between concepts as considered in the matrix presented in Figure 61, we defined four types of relations validation:

- **Architectural Usage**. Underlined correct sign (✓). GOALS architectural relations. Define relations between components which are generated by means of the application of the GOALS method, and which are part of its meta-model, as presented throughout Chapters 3. The Goals Language. It's the "inference model".
- **Allowed Usage**. Correct sign (✓). Relations that can be applied for the purpose of architectural optimization. Mostly represent: *Business Modeling Flexibility* (✓1), in order to allow other possible representation such as the "Usage of Spaces (IXs) by BPs", or the "Direct Usage of DE by IX"; or *Software Architectural Horizontal Reuse* (✓2) of the concept by itself, of to use other concepts from the same layer e.g. any of the Software Architecture concepts invoking itself (notice that the UISR and BDSR concepts are structurally equal, reason why the ✓2 validation also applies in this case).

- **Contingency Usage**. Wrong sign (×). Which are relations that that should not occur, but may represent a useful trade-off, as they can simplify implementation, however introducing architectural disorganization: *Business Architectural Restrictions* (×1), as the Enterprise Structure BRs should always be accessed by IXs and not directly, and also as DEs should be accessed by means of UI or DB SRs, and not by User Interface components; *Software Architectural Restrictions* (×2), in order to ensure the architectural control as a result of the one-to-one relation of the User Task to the AS, of its User Intentions to the IC, and of its User Interactions to the IO, from the two-step application of the *Software Architectural Technique*.
- **Restricted Usage**. Wrong underlined sign (×). Relations that should not exist, as the BP and UT concepts cannot directly use Business Logic software-specific concepts, the IX should not use any software-specific component, and the BR should not use any other BR, in order to promote their independence in terms of *regulations* validation, unless that criteria is introduced in the architectural specification for purposes of improvement of the framework.

The **Architectural Usage** relations represent the "solution space", which is the one in which we now focus in order to sustain our research hypothesis. While the validation by means of the CCA method supports the first part of H1: "*It is possible by means of a cross-consistent relation of business and software concepts*", the application of the method validates the second part: "*and by modeling the business human activity and relating it to software-specific components* […] *by means of the application of architectural patterns*". Figure 62 presents the CCM "inference model" of the GOALS language and its relation with the Steps of the method. Enterprise Structure concepts crossing relations are presented in darker doted green, and software-specific concepts relations in lighter plain green, the invocation from business to software or the opposite is presented in lighter doted green.

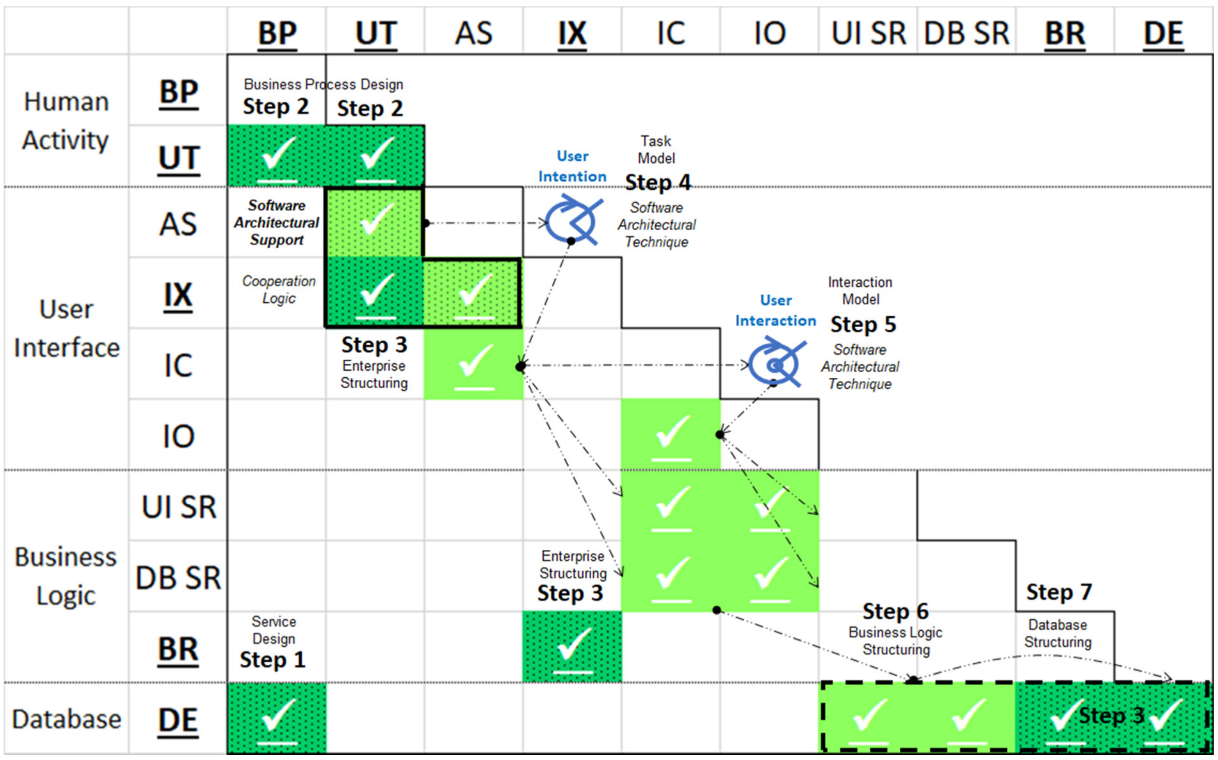


Figure 62. Cross-Consistency Matrix and the GOALS Method Steps.

Thus, the second part of H1, which states that the cross-consistency should be complemented with a method that applies the language, is sustained by the following techniques:

- The *Software Architectural Support* (UT-AS crossing), as a one-to-one relation of the business-specific User Task (UT) concept and the software-specific Aggregation Space (AS), that ensures support for a complete human (user) with one User Interface. This structural relation is maintained thorough Steps 2 to 4. After it, the AS is decomposed into ICs and IOs using independent techniques in the following Steps.
- The *Cooperation Logic* (UT-IX and AS-IX crossings), that ensures that the same IX support is given to the pair UT-AS one-to-one relation, ensuring that the same *regulations* (BRs) and information (DEs) are provided, independently of a physical of computerized IX.
    - This provides the isolation of the business responsibility which is enclosed in business *regulations* in terms of BRs (Business Rules), as the business-specific concept of the Business Logic, among software-specific concepts.
    - This also allows the elaboration of the AS as a result of the IX to which its own UT is attached, increasing the probability of finding reusable structures of User Interface, that will serve more than one AS by means of recognizing the *business concept* (DE) being managed.
    - By means of recognizing reusable User Interface components, the recognition of the Business Logic components may also be facilitated by means of recognizing which UISR or DBSR components may provide the needed *data* for the User Interface component (IC or IO) or any of its supporting System Responsibilities (SR).
- The *Software Architectural Technique*, that by means of the consecutive decomposition of the UT in terms of User Intentions and User Interactions (steps 4 and 5), results in the elicitation of ICs and IOs that use the corresponding UISR and BDSR (which use Data Entity elements), in order to elaborate the internal structure of the Information System.
- The Business Logic Structuring (Step 6), that applies semiotic analysis based on the identified *business concepts* (DEs) in order to relate the Business Logic components to the Database Tables and Fields, and that isolates the responsibility of BRs and the business-specific component of this layer.
- The Database Structuring (Step 7), that by means of semiotic analysis elaborates the *data* structure that reflects the existing enterprise *business concepts*, composed in a single diagram (the UML Domain Model) of DEs and corresponding Fields.
- The Software Architecture (Step 8), is simply the presentation of every concept's component and their relations (DEs relations are usually omitted, and the UT-IX relation is replaced by the UT-AS one-to-one relation in the cases when a computerized support exists).

Finding that the **Architectural Usage** relations presented in the CCM are valid, and that the GOALS techniques that elicit and generate each component are suitable for the specification of a Software Architecture for a given BPI, independently of technology used for implementation, in order to provide support for a BPI, which *business process* is designed or may be translated to the GOALS language, we sustain our hypothesis.

6.2. Power of Expression

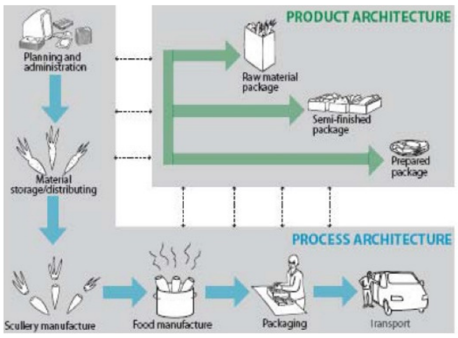
The empirical validation of the power of expression of our method targets understanding how GOALS may be used to model different types of business and software development aspects, in the perspective of the identification of *requirements* and the (model-driven) transformation of these *requirements* in software architectures. The methods analyzed in Chapter 2. State of The Art, are now compared in terms of the used techniques and their scope concerning the business and software modeling. The models considered for comparison are relative to: the social organization of the human activity in *business processes* and services, the correspondent derivation of *requirements*; the *task* model, which specifies the organization of the individual human activity in order to complete a *task*; the conceptual and implementation-independent software structure for of the User Interface, Business Logic, and Database of the Information System. Each method concepts are mapped to GOALS concepts in order to evaluate: comprehensiveness in large scope and in detail; *requirements* elicitation; and software architecting techniques, in every case that the comparison applies. The differences and similarities are highlighted, and qualitative evaluations and modeling suggestions are made.

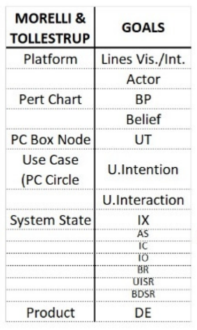
6.2.1. Service Design

The Service Design develops a "high-level" perspective of the functioning of the enterprise. In this perspective, the relation of *business processes* and what they produce and consume provide a view of the structure of *goals* of the enterprise, their internal and external *actors*.

MORELLI & TOLLESTRUP

Figure 63 presents a model of MORELLI & TOLLESTRUP [Morelli & Tollestrup, 2006], which represents a large fabric of food. At the enterprise modeling level, the *business processes* granularity is not adequate to represent large groups of *business processes* which are related in order to produce complex products, reason why they are represented by "Platforms" (right side).





MORELLI & TOLLESTRUP	GOALS
Platform	Lines Vis./Int.
	Actor
Pert Chart	BP
	Belief
PC Box Node	UT
Use Case (PC Circle	U.Intention
	U.Interaction
System State	IX
	AS
	IC
	IO
	BR
	UISR
	BDSR
Product	DE

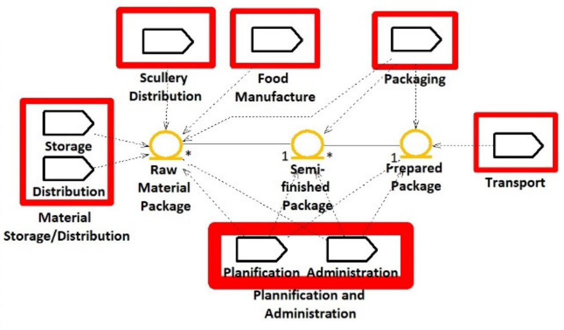


Figure 63. MORELLI & TOLLESTRUP and GOALS Service Design Representation.

In Figure 63, the fabric of food has a sequence of platforms (e.g. "Food Manufacture"), which define the process of production. In GOALS, the platforms are matched by the Interaction and Visibility Lines (only the last is represented) and the DEs that they manage, which are common to other platforms. In GOALS, the relation between Business Processes (BPs) does not exist (reason why the BPs access directly to the DEs, instead of the platform), because the flow of information is analyzed by means of the structure of Business Rules (BRs) for *data* validation during the execution of the BP. Nevertheless, the execution may be analyzed by means of the cardinality of the relation between DEs e.g. one "Prepared Package" has many "Semi-Finished Package", and also by means of the semantical analysis of the involved *concepts*. MORELLI & TOLLESTRUP identify use cases but not the internal Software Architecture.

SERVICE EXPERIENCE BLUEPRINT (SEB)

The BLUEPRINT is a reference in the modeling of the interaction of the customer with the enterprise given a certain business perspective. It defines the line of interaction and the lines of visibility, which are important, inclusively in a marketing perspective, specifying what the customer should see or not. The SERVICE EXPERIENCE BLUEPRINT (SEB) [Patrício et al., 2008] is a based on the BLUEPRINT, and targets Information System development using a logic of decomposition compatible with *essential use cases* in terms of the generated requirements granularity. The method contributes to clarify the relation of the customer with the system, while predicting that it has a “back-office” system, beyond the line of visibility.

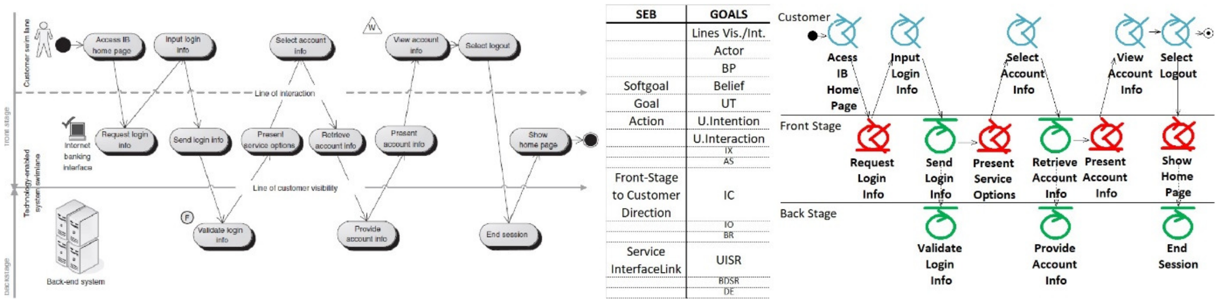


Figure 64. SEB and GOALS Comparison.

In the example presented in Figure 64, a customer-service interaction identified by means of a *goal* (mapped to a User Task, a *goal* within the service to the customer), is decomposed according to the interaction of the customer with the front-end of the service. In the cases when there is an arrow in the direction of the customer *task*, mapped to a User Intention in GOALS, the information is presented using an Interaction Component (an IC). And if the direction is towards the front-end, it is received by a User Interface System Responsibility (UISR), which accesses a back-stage of Service Interface Links (also represented with UISRs).

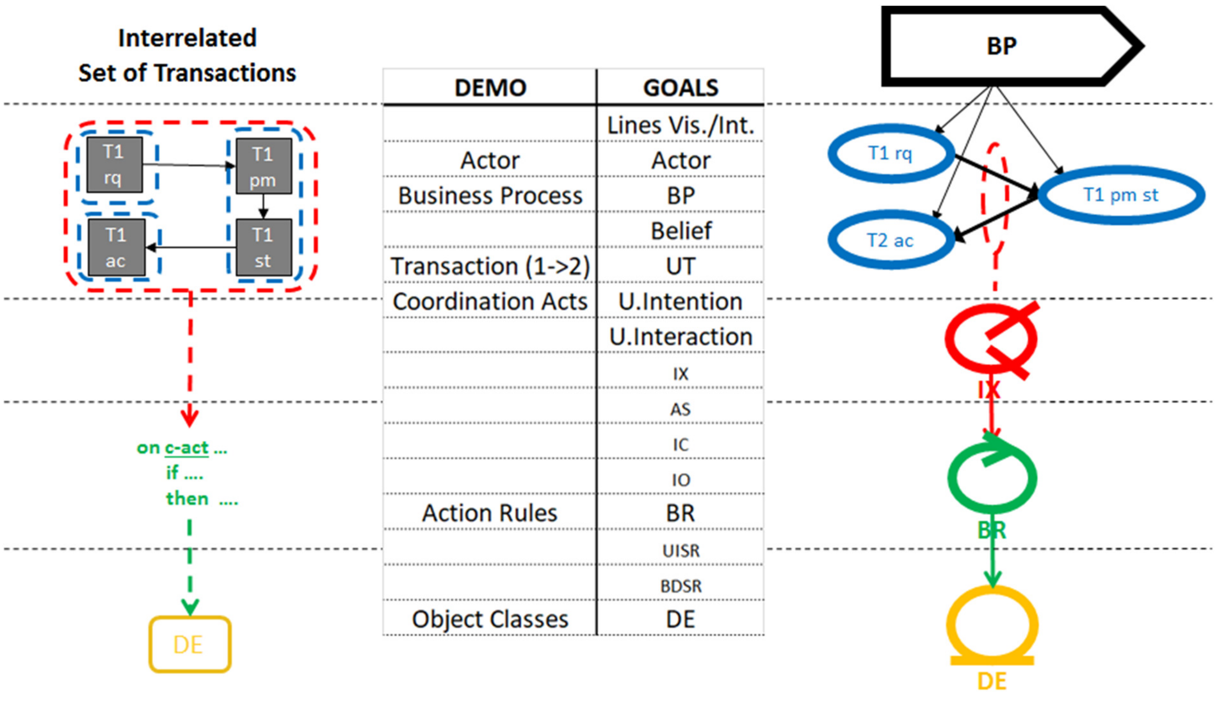
The multi-interface modeling proposed by SEB could be based on the decomposition of the User Intention and ICs into User Interactions and Interaction Objects (IO) and structured into Aggregations Spaces (concerning each UT), implementing Steps 2 (BP design), 4 (Task Model) and 5 (Interaction Model) of the GOALS method, to reach a structured User Interface design. This possibility may enhance implementation time, as in most cases, the difference between the user performing the *tasks* alone in his computer, and with a bank collaborator, in person, or remotely, is that the both will use distinct implementation of each tool, and a different set of permissions which may be described in the Interaction Model (Step 5).

6.2.2. Enterprise Engineering

DEMO [Dietz, 2006], is the reference method in EE. It uses *transactions* in order to organize the enterprise human activity, and produces models which contain information that may be used to structure the Business Logic and the Database of an Information System. ARCHIMATE, the BUSINESS MOTIVATION MODEL, and ARMOR are also compared with GOALS in terms of common concepts identification.

DEMO

The GOALS structure may be compared to the DEMO structure based on the analysis of the concepts which are involved in the *transactions* of the *business processes*. The GOALS Business Process (BP), User Task (UT), Business Rule (BR) and Data Entity (DE) concepts are compatible and can be derived from the DEMO concepts of: (Transaction) Coordination Act (C-Act), Action Rule and Object Class, respectively. GOALS adds to those concepts the *Goal* (name) of the Business Process (BP), and the Interaction Space (IX). For the purpose of understanding how GOALS provides support for the enterprise human activity, we establish a relation between the "inter-related sets of *transactions*", which define a BP in DEMO, with a BP in GOALS.



DEMO	GOALS
	Lines Vis./Int.
Actor	Actor
Business Process	BP
	Belief
Transaction (1->2)	UT
Coordination Acts	U.Intention
	U.Interaction
	IX
	AS
	IC
	IO
Action Rules	BR
	UISR
	BDSR
Object Classes	DE

Figure 65. DEMO and GOALS Relation of Concepts.

Figure 65 presents the relation that is established between the DEMO *transactions* and the GOALS BP and its UTs. UTs are related to DEMO *transactions* by means of merging consecutive Coordination Acts (C-Acts). The C-Acts are: request (rq), promise (pm), state (st), and accept (ac). User Tasks (UT) are derived from DEMO in terms of consecutive C-Acts performed by the same *actor* (e.g. sequence "T1 pm st"). In DEMO the *transaction* activity is related to Actions Rules, in the same way that in GOALS the UTs are related to the space (IX) where they occur, and then related to BRs, being therefore mapped to Actions Rules (from which their definition is derived). The Action Rules are then related to Class Objects in DEMO, in the same way that BRs are related to DEs in GOALS.

The identification of IX is based on the space used by *actors* in order to perform their UTs, and the relation with the concept of Business Rule is based on the WISDOM *Software Architectural Technique* which relates human activity (the UTs) with system behavior (the BRs) by means of an IX. Hence, it serves as a door that relates business-and-software recognizable concepts of Interaction Space, Business Rule and Data Entity. GOALS uses the concept of IX, in order to establish a relation with the User Interface and remaining structure of the Information System (the Business Logic and the Database). The DEMO method does not consider the human user activity, as every *transaction* only specify two interactions between two users, with no changes. As a result, the resulting systems need adjustment in terms of User Interface design based on other methods, reason why we believe that the IX is an important increment to DEMO concerning the elaboration of the business model.

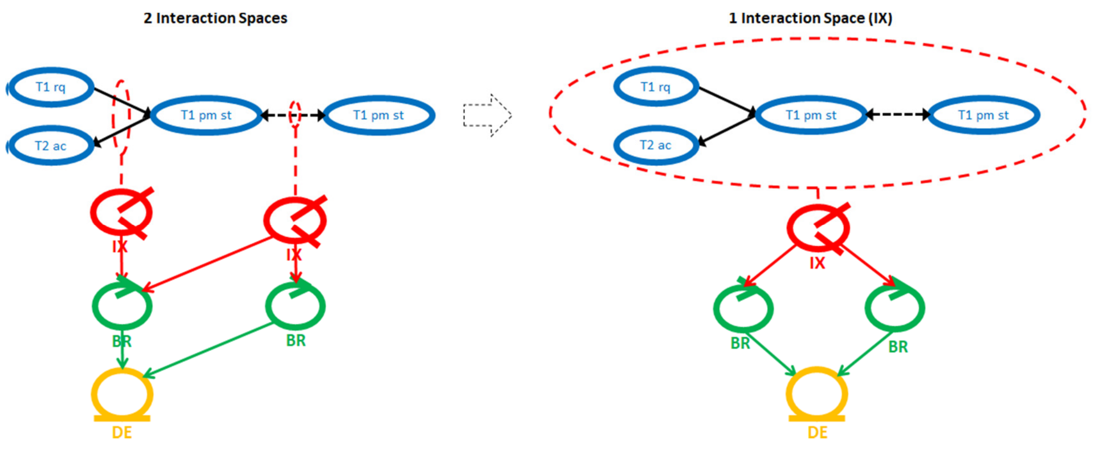
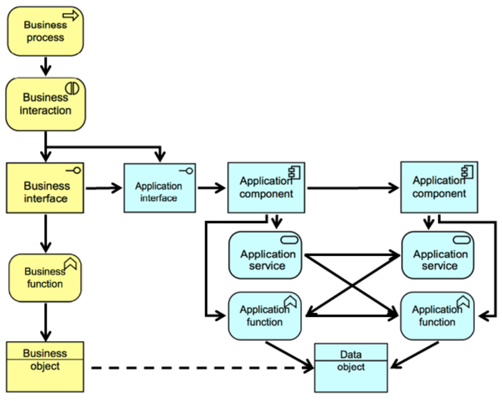


Figure 66. GOALS Representation of *Transactions* including More Than One Actor.

The establishment of the IX has the benefit of also allowing the specification of different types of relations of human activities, namely the bi-directional relation (on the right side of the structures presented in Figure 66), in this case presented using a Conditional relation (dashed bi-directional line), equivalent to a flow of three UTs between two actors, in this case represented with a Sequential relation (two full one-direction line, on the left of the structures). The Conditional relation means that the user may not perform a UT, and that the BP execution will continue as long as existing BRs are respected. When a bi-directional Conditional relation is used, at least one BR should be shared between the related UTs, meaning that there is a common aspect to be treated by the *actors*. Figure 66 illustrates the situation and also the transformation of a Sequential and Conditional BP execution first supported by two IXs (on the left) to one IX (on the right), meaning that all the *actors* (in this case three *actors*) perform their UTs in the same physical or logical space. Thus, the difference between both methods business conceptual structures is that GOALS considers the human interaction within a certain space besides the remaining conceptual structure. This creates conditions to further consider relevant aspects such as the *actor* context, the physical and logical space that they use, and the UTs which they carry on. The relation of derivation of IXs from UTs may be specified for the support of more than two UTs, and ultimately for the support of all of the UTs of a BP.

ARCHIMATE

The ARCHIMATE Language models the business and the software parts of the architecture of an enterprise, and is oriented for the identification of the building blocks of the enterprise. We used the concepts provided by the language in order to identify the possible mappings between both languages so that the GOALS language could be represented using ARCHIMATE.



ARCHIMATE	GOALS
Business Service	Lines Vis./Int.
Business Actor Business Role	Actor
Business Process	BP
	Belief
Business Interaction	UT
	U.Intention
	U.Interaction
Business Interface Location	IX
Application Interface	AS
	IC
	IO
Business Function	BR
Application Service	UISR
Application Function	BDSR
Data Object Business Object	DE

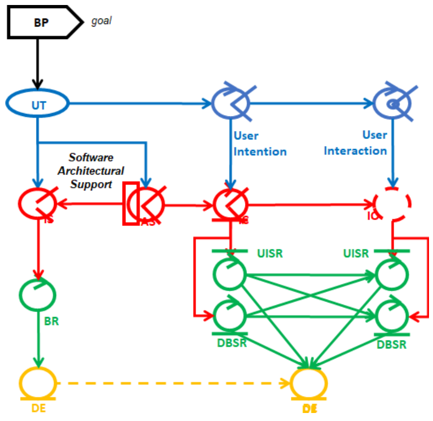
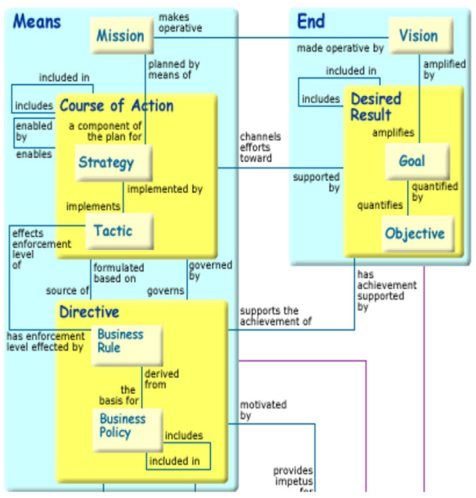


Figure 67. ARCHIMATE Modeling of the GOALS Structure.

Figure 67 presents the relation of concepts of ARCHIMATE and GOALS. The UT is mapped to the “Business Interaction” reflecting the interaction between *actors* that occurs in GOALS each performing its UT. In the business perspective, the “Business Interface” and the “Location” concepts are compliant with the notion of IX, and the “Business Function” is compliant with the BR, and the “Business Object” with the DE. In the software perspective, the “Application Interface” is compliant with the AS, and since we do not find specific and suitable representation to the IC and IO components, we mapped these concepts to the generic “Application Component”. The UISR and BDSR are mapped to the “Application Service” and “Application Function” concepts. ARCHIMATE distinguishes the “Business Object” from “Data Object”, whereas we map both to the DE representation.

BUSINESS MOTIVATION MODEL

The BUSINESS MOTIVATION MODEL (BMM) is more closely related to GOALS in terms of the *ideology* specification of the enterprise, than in the perspective of the operational implementation of the enterprise decisions. The “Strategy” and “Tactics” reveal enterprise concepts which may be applied as guiding lines in the specification of *business processes*, which is work in progress in GOALS.



BMM	GOALS
	L. Visib.
	L. Int.
Goal	*goal*
Objective	BP
Mission	*Ideology*
Vision	*Belief*
Business Policy	Principles
Strategy/Tactic	UT
Business Rule	BR

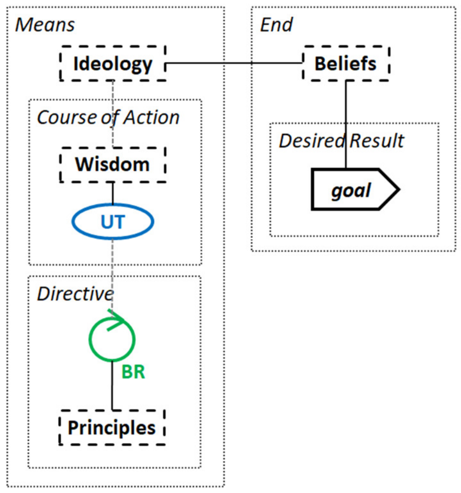
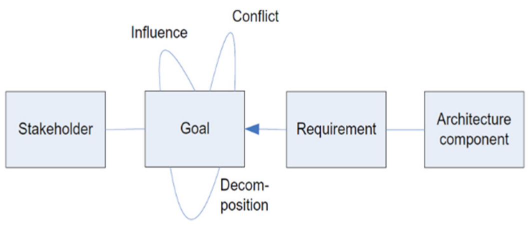


Figure 68. BMM Model using the GOALS Language.

The higher level concepts used in the BMM are presented in Figure 68, which define a “Mission” (*ideology* on GOALS) from which is established a “Vision” (*beliefs*) which is specified by means of *goals* (BP and its *goal*) and “Objectives” (quantifiable “*goals*”), which are supported by a “Course of Action” which “Strategy” (*wisdom*) and “Tactics” (UT) are governed by “business rules” (BR) and “business policy” (*principles*). Both approaches are similar in the way that they relate the concepts, yet, distinct in the form that they organized them. The GOALS existing relations are presented with a full line, and the non-existent with a dashed lighter line.

ARMOR

ARMOR produced a simplified Enterprise Architecture which has a similarity with GOALS in terms of the (one-to-one) relation that is established between the business requirement and its implementation. It supports modeling the *goals* of the stakeholders and their decomposition, including “influence” and “conflict” control, from which it withdraws requirements which should be supported by an “architectural component”. The relation is presented in Figure 69.



ARMOR	GOALS
Stakeholder	Actor
Requirement	User Story
Hard Goal	UT
Soft Goal	U. Intention
Architecture Component	AS

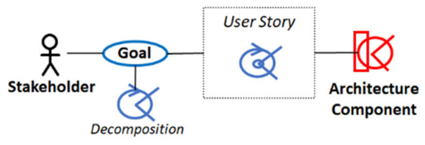


Figure 69. ARMOR Structure and Representation using the GOALS Language.

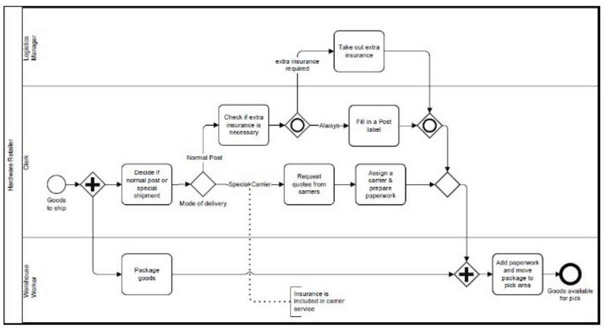
The mapping between ARMOR and GOALS is coincident in the relation that is established by "Hard Goals" (quantifiable *goals*) and the ("complete") UTs, and the "one-to-one" relation of the "Architecture Component" with the AS (or any other User Interface component), in order to provide the same support. ARMOR is another example of the compatibility of GOALS and the *goal*-oriented techniques which motivates future work regarding the evolution of GOALS concerning the specification of the UX *requirements*, and further development of the Task Model and software architectural reuse improvements.

6.2.3. Business Process Management

BPMN is the reference modeling language in BPM. It may be used to model human activity, *resources* and rules, however without specifying a method.

BUSINESS PROCESS MODEL AND NOTATION (BPMN)

BPMN provides a very consistent representation of *business process,* that nevertheless allows that one user may carry on more than one *task* consecutively, an aspect that is not benevolent for Information Systems development, as it may lead to the identification of more *use cases* then needed, which may be reflected in more components depending on the patterns used to elaborate the Software Architecture. Also, the representation of *business processes* allows the representation of part of the business *regulations* in the *business process* design, which we do not observe as a good practice, since these *regulations* should be independent from the *business process* execution, otherwise, slight *regulations* changes may change its design, and as such lead to the loss of traceability.


Logistics Manager
Hardware Retailer
Clerk
Warehouse Worker
Take out extra insurance
extra insurance required
Check if extra insurance is necessary
Always
Fill in a Post label
Normal Post
Goods to ship
Decide if normal post or special shipment
Mode of delivery
Special Carrier
Request quotes from carriers
Assign a carrier & prepare paperwork
Package goods
Insurance is included in carrier service
Add paperwork and move package to pick area
Goods available for pick

BPMN	GOALS
	Lines Vis./Int.
	Actor
BPMN	BP
	Belief
+ Activities	UT
Activity	U.Intention
	U.Interaction
	IX
	AS
	IC
	IO
Gateway	BR
	UISR
	BDSR
	DE

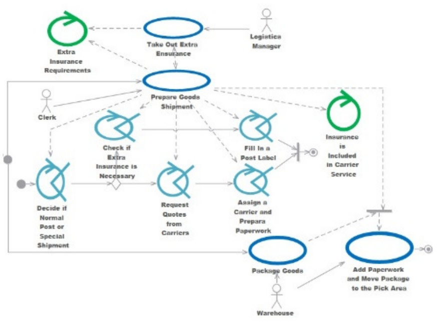
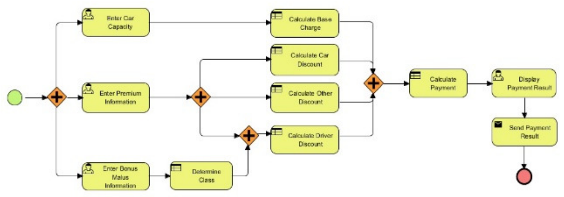

Extra Insurance Requirements
Take Out Extra Insurance
Logistics Manager
Prepare Goods Shipment
Clerk
Check if Extra Insurance is Necessary
Fill in a Post Label
Insurance is Included in Carrier Service
Decide if Normal Post or Special Shipment
Request Quotes from Carriers
Assign a Carrier and Prepare Paperwork
Package Goods
Add Paperwork and Move Package to the Pick Area
Warehouse

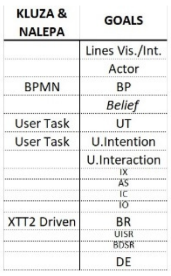
Figure 70. BPMN and GOALS equivalent BP Design.

Taking as an example the BPMN "Shipment Process of a Hardware Retailer" diagram presented in Figure 70, the "Check if Extra Insurance is Necessary" activity may depend on a business *regulations* concerning "insurances" which is only expressed by means of the human activity and the flow of the *business process*, and is never expressed in the resulting structure as an independent *business concept*. If the *regulation* concerning insurance may be applied to other *business processes*, and if it changes, then the design of both may have to change, with even a bigger impact in the supporting Information System.

KLUZA & NALEPA

In our perspective, the KLUZA & NALEPA [Kluza & Nalepa, 2013] method presents a solution for the problem of rule representation using BPMN, by means of a semantic analysis which allows the isolation and separation of business rules concerns, producing information to structure the Business Logic of the Information System.





KLUZA & NALEPA	GOALS
	Lines Vis./Int.
	Actor
BPMN	BP
	Belief
User Task	UT
User Task	U.Intention
	U.Interaction
	IX
	AS
	IC
	IO
XTT2 Driven	BR
	UISR
	BDSR
	DE

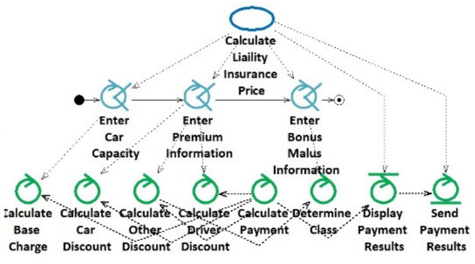
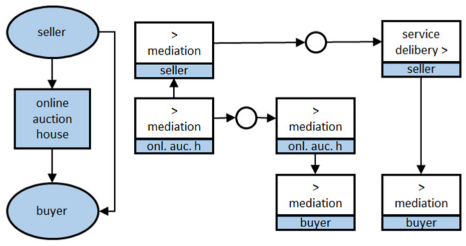


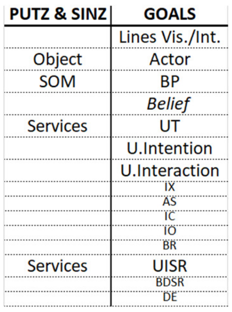
Figure 71. KLUZA & NALEPA and GOALS equivalent Requirements Elicitation.

Figure 71 presents the BPMN and Business Rules combination (adapted example diagram) produced by the method, and the equivalent GOALS representation, which identifies the BPMN activities as User Intentions, once they are performed by the same user in the "Calculate Liability Insurance Price" UT. From the BPMN diagram it is possible to extract the dependency of the User Intentions from the business rules, and also the dependency between them, and also the transition to the "Display Payment Results" and the "Send Payment Results", which may be mapped to UISR and DBSR components, respectively.

PUTZ & SINZ

PUTZ & SINZ [Pütz & Sinz, 2010] present a solution that enables the identification of channels of communication by means of business objects. The method applies the SEMANTIC OBJECT MODEL (SOM) and identifies *transactions* and *tasks* by means of the interaction schema (IAS) and the *task*-event schema (TES), then producing decompositions of the system until reaching an executable BPMN description. This information may be useful in order to identify the *tasks* and the User Interface spaces in a methodical way, however, it lacks a domain model, which could benefit the approach. The model and an example are presented in Figure 72.





PUTZ & SINZ	GOALS
	Lines Vis./Int.
Object	Actor
SOM	BP
	Belief
Services	UT
	U.Intention
	U.Interaction
	IX
	AS
	IC
	IO
	BR
Services	UISR
	BDSR
	DE

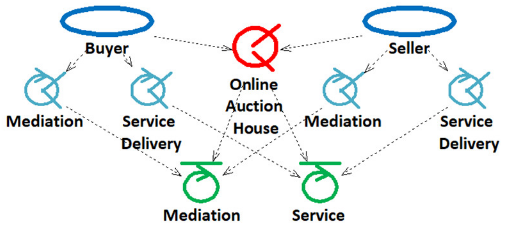


Figure 72. PUTZ & SINZ and GOALS equivalent Representation.

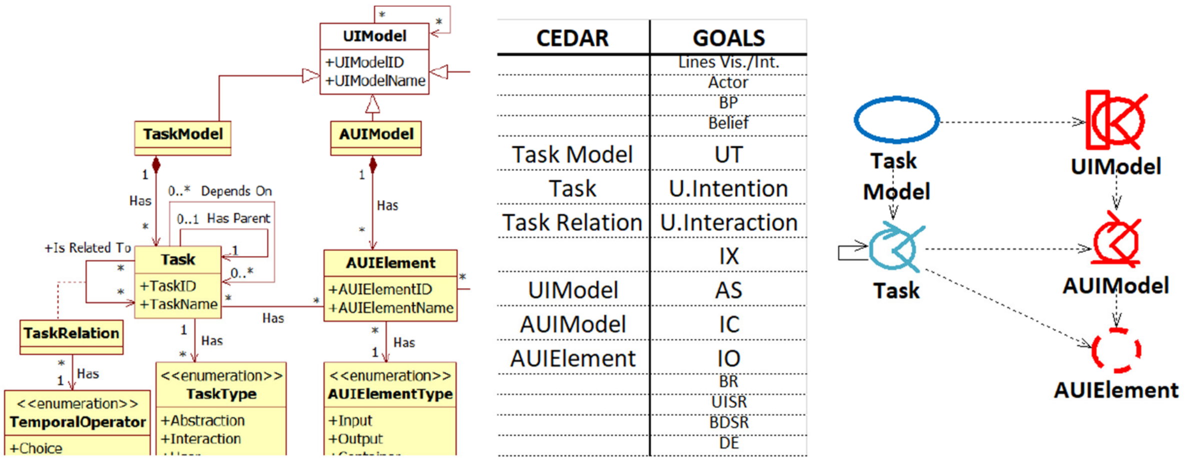
6.2.4. Human-Computer Interaction

The Human-Computer Interaction (HCI) methods produce software architectures to respond to the user interaction providing adequate User Interface behavior, using User-Centered Design (UCD) techniques. The architectures are usually complex, as they structure the spaces of the User Interface, and its fields in terms of the type, usually the managed *data* concepts, and also the behavior that should be applied in terms of relations between fields and spaces.

The combined HCI and SE are recognized as Human-Centered Software Engineering (HCSE) methods, in which WISDOM was pioneer by means of the *Software Architectural Technique* (as referred in the thesis) generating the User Interface and Business Logic of the Information System, further complementing it with the Domain Model, completing the MVC architecture. Therefore, the question related to HCI does not concern the Software Architecture elaboration only, but to how representative the method may be of the business in order to provide support for a BP, elicit *requirements* and maintain traceability.

CEDAR

The CEDAR method [Akiki, 2013] establishes a relation between a *task* model (of e.g. a *use case*), decomposing it until the physical interaction between human and system (e.g. a click), and relates this activity to User Interface components. The structure is presented in Figure 73, where the CEDAR method example architecture is modeled using GOALS. The final structure is similar to the GOALS User Interface model, as the *use case* (equivalent to "TaskModel") is related to the User Interface specification ("AUIModel"), and the "Task" is related to the "AUIElement", to which a certain type of interaction object is specified ("AUIElementType").

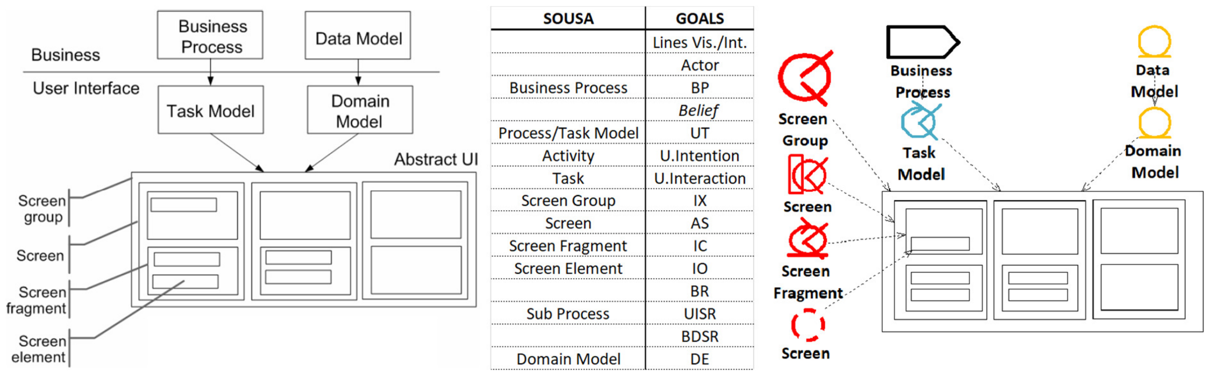


CEDAR	GOALS
	Lines Vis./Int.
	Actor
	BP
	Belief
Task Model	UT
Task	U.Intention
Task Relation	U.Interaction
	IX
UIModel	AS
AUIModel	IC
AUIElement	IO
	BR
	UISR
	BDSR
	DE

Figure 73. CEDAR and GOALS HCI Representation and User Interface Architecture.

CEDAR develops the type of User Interface architecture that enables complex specifications that support *requirements* specified by User Stories [Memmel et al., 2007], usually more detailed than *use cases* in terms of interaction specifications and system behavior, which may be complemented with scenarios that may express distinct contexts according e.g. the elaboration of personas. The CEDAR method is however, not representative of the business.

SOUSA



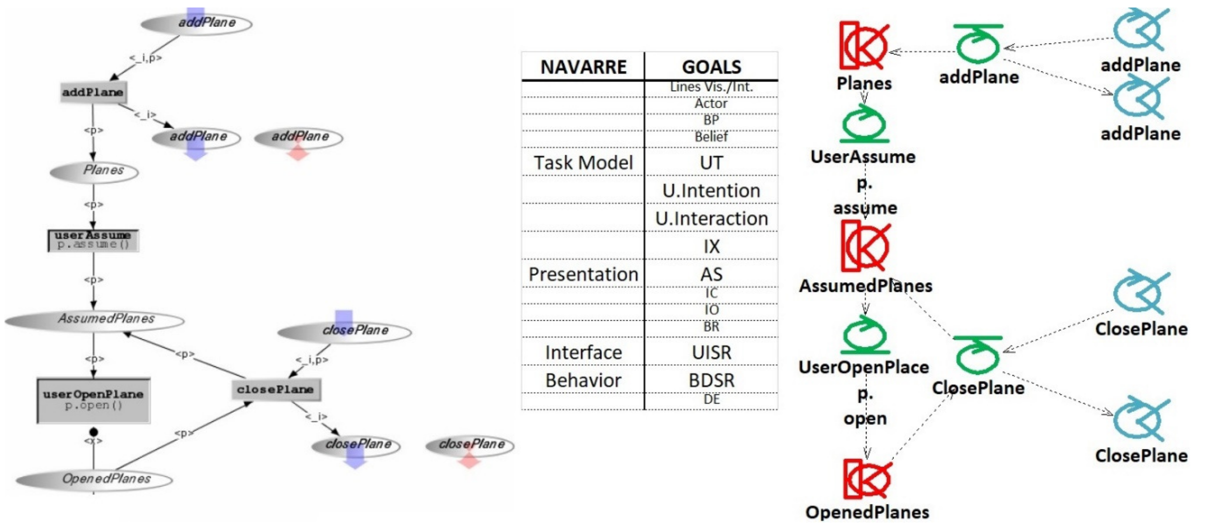
SOUSA	GOALS
	Lines Vis./Int.
	Actor
Business Process	BP
	Belief
Process/Task Model	UT
Activity	U.Intention
Task	U.Interaction
Screen Group	IX
Screen	AS
Screen Fragment	IC
Screen Element	IO
	BR
Sub Process	UISR
	BDSR
Domain Model	DE

Figure 74. SOUSA and GOALS User Interface Architectural Similarities.

The SOUSA method [Sousa et al., 2008], establishes a relation between *business processes* and *data* by means of a *task* model, from which it derives a User Interface which has an equivalent structure to the GOALS User Interface structural logic. SOUSA structures a "Screen", a "Screen Fragment", and a "Screen Element" similarly to GOALS: Aggregation Space, Interaction Component and Interaction Object. Furthermore, a "Screen Group", defined as "a group of closely related screens" can also be related to the AS of a given Interaction Space (IX). This is the level of complexity which we find that is acceptable for the development of the User Interface of an enterprise Information System, since this structure provides enough detail in order to group complete tasks (User Tasks), User Intentions and Interactions.

NAVARRE

There are other types of architectures, which are more specific in terms of the rigor which is architecturally introduced by its structure. In the case of the NAVARRE method [Navarre et al., 2009], actions and the User Interface architectural components are related by means of a sequence of human and system execution. At this level of specification, the feedback is also structured by means of a rigorous structure in which attention is also paid to the parameters specifying the flow of information (Figure 75).



NAVARRE	GOALS
	Lines Vis./Int.
	Actor
	BP
	Belief
Task Model	UT
	U.Intention
	U.Interaction
	IX
Presentation	AS
	IC
	IO
	BR
Interface	UISR
Behavior	BDSR
	DE

Figure 75. NAVARRE and GOALS User Interface Architecture Representation.

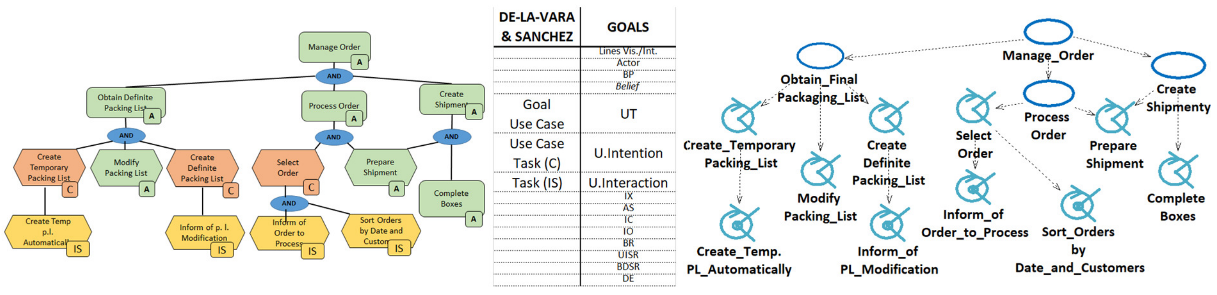
This reinforced structure of information also specified more detailed and probably better well-built User Interfaces since this level of detail will augment the probability of identifying errors sooner and predict its behavior more clearly. However, this level of detail is out of the scope of the present study.

6.2.5. Requirements Engineering

The Requirements Engineering (RE) methods are considered in the perspective of goal, use case, and combined elicitation techniques, and their relation with software specification.

GONZÁLEZ & DÍAZ

The GONZÁLEZ & DÍAZ method [González & Díaz, 2007] is a *goal*-driven approach that decomposes *goals* until the *task* level, from which it derives *use cases* as *requirements*. It is an effective logic of decomposition, which however in our perspective has the problem of eliciting *use cases* from different levels of granularity, which may correspond to GOALS User Tasks or User Intentions, creating indefiniteness in terms of the structuring of the Information System.


Manage Order
A
AND
Obtain Definite Packing List
Process Order
Create Shipment
Create Temporary Packing List
C
Modify Packing List
Create Definite Packing List
Select Order
Prepare Shipment
Complete Boxes
Create Temp p.l. Automatically
IS
Inform of p. l. Modification
Inform of Order to Process
Sort Orders by Date and Customers
Manage_Order
Obtain_Final Packaging_List
Process Order
Create Shipmenty
Create_Temporary Packing_List
Create Definite Packing_List
Select Order
Prepare Shipment
Modify Packing_List
Inform_of Order_to_Process
Complete Boxes
Create_Temp. PL_Automatically
Inform_of PL_Modification
Sort_Orders by Date_and_Customers

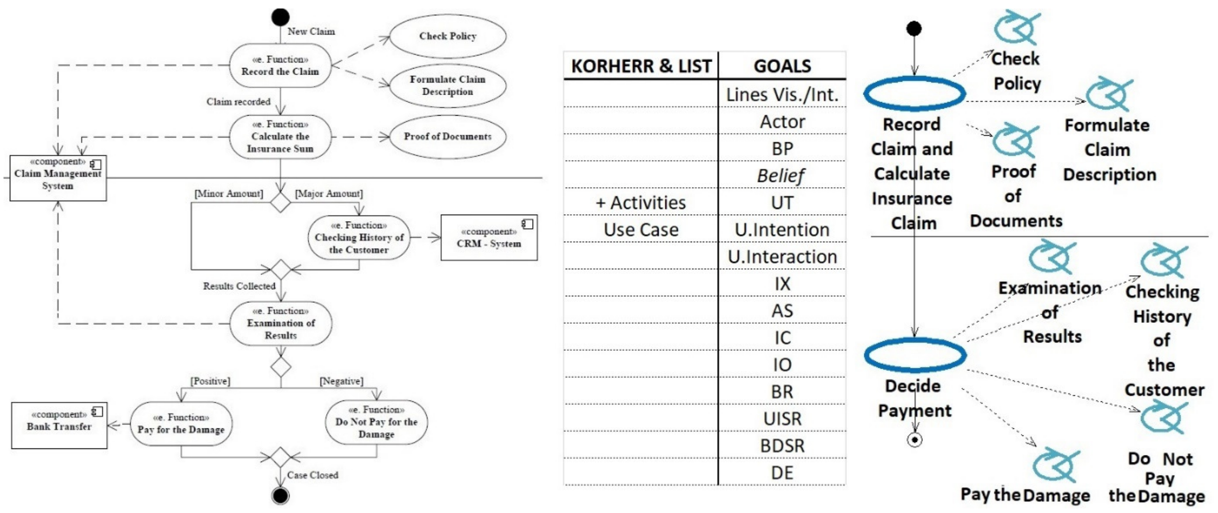
DE-LA-VARA & SANCHEZ	GOALS
	Lines Vis./Int.
	Actor
	BP
	Belief
Goal Use Case	UT
Use Case Task (C)	U.Intention
Task (IS)	U.Interaction
	IX
	AS
	IC
	IO
	BR
	UISR
	BDSR
	DE

Figure 76. GONZÁLEZ & DÍAZ and GOALS Requirements Elicitation.

Figure 76 presents the method and what would be the equivalent representation with the GOALS language, from which it is possible to establish a relation between the *use cases* and User Tasks (UT), and the method *tasks* and User Intentions, in which the indefiniteness of use cases is expressed by the first and second levels of *use cases* which are identified using the GOALS language.

KORHERR & LIST

The KORHERR & LIST method [Korherr & List, 2006], captures *use cases* from *business process* models in what is one example why the alignment of the system implementation and the *business process* design traceability may be lost when the system is implemented. We named this situation as *use cases* "Indefiniteness", when it is possible to derive a different number of *use cases* from the same process, which would result in different software conception, probably with different software architectures.


New Claim
«e. Function» Record the Claim
Check Policy
Formulate Claim Description
Claim recorded
«e. Function» Calculate the Insurance Sum
Proof of Documents
«component» Claim Management System
[Minor Amount]
[Major Amount]
«e. Function» Checking History of the Customer
«component» CRM - System
Results Collected
«e. Function» Examination of Results
[Positive]
[Negative]
«component» Bank Transfer
«e. Function» Pay for the Damage
«e. Function» Do Not Pay for the Damage
Case Closed
Check Policy
Record Claim and Calculate Insurance Claim
Formulate Claim Description
Proof of Documents
Examination of Results
Checking History of the Customer
Decide Payment
Do Not Pay the Damage
Pay the Damage

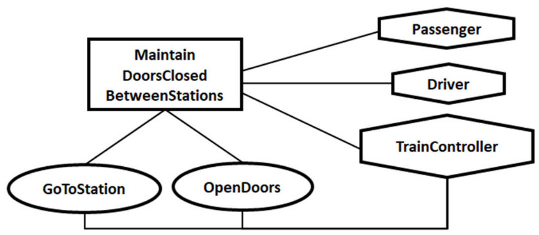
KORHERR & LIST	GOALS
	Lines Vis./Int.
	Actor
	BP
	Belief
+ Activities	UT
Use Case	U.Intention
	U.Interaction
	IX
	AS
	IC
	IO
	BR
	UISR
	BDSR
	DE

Figure 77. KORHERR & LIST and GOALS Requirements Elicitation Representation.

Figure 77 presents the same *business process* modeled by KORHERR & LIST and GOALS, in which the activities performed by the same *actor* e.g. "Record the Claim" and "Calculate the Insurance Claim" are merged in the same, a UT named as "Record the Claim and Calculate the Insurance Claim", which includes "Check Policy", "Formulate Claim Description" and "Proof of Documents" as User Intentions. The difference between the two methods is that GOALS separates the concerns targeting software implementation, promoting the simplification of the implementation. The actions of the *actor* are performed over a User Interface and inevitably the rules are implemented in programmed code, and by separating the concerns, less User Interfaces will be implemented, and also the dependency of their logic with the rules will be diminished.

KAOS

KAOS [Lamsweerde, 2000], is *goal*-oriented method that relates human *goals* and *tasks* to system functionality, which vocation is the elaboration of software systems, but not necessarily Information Systems, as the functionality specification is not supported by a Domain Model. KAOS specifies "*Goals*" and their decomposition (including concurrency), the "Agents" and "Objects", active and inactive system components, respectively, and the "Requirements" and "Expectations" which are system and human responsibilities, also respectively.



KAOS	GOALS
Agent	IX
Goal	U. Intention
Object	DE
Action	UISR

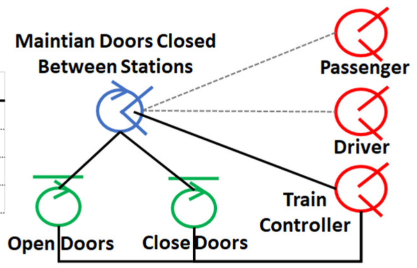
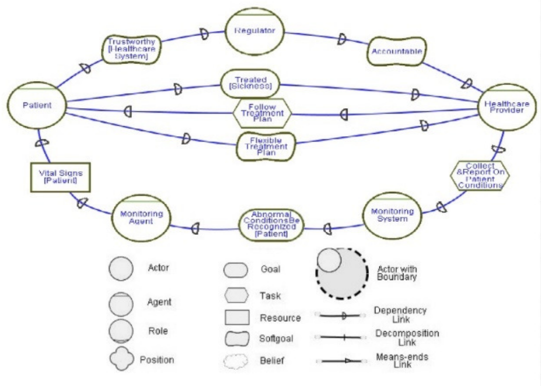


Figure 78. KAOS and GOALS Concept Mapping and Example.

Figure 78 presents a KAOS (adapted) diagram and its modeling using the GOALS language, in which, the *agent* is represented by Interaction Spaces (IX), which may be considered as equivalent in terms of ensuring regulations and data management. The goal is mapped to the User Intention, and these to System Responsibilities, which are equivalent to KAOS "actions". By means of maintaining a structure relation between the business requirements, derived from their expectation, the KAOS method ensures traceability between business and the software up to the specification of the software *agent*, however, not the User Interface or the Database.

I-STAR (i*)

The I* (i-star, or i*) modeling framework [Yu, 1995], defines a set of concepts, namely *actor*/agent, *task*, *goal*, *softgoal* and *resource*, which are very similar to GOALS in the way that they model the interaction between the human *actors*. Both see the communication between *actors* as a sequence from one *actor* to the other by means of sharing something, which in most cases is a *task*, a *goal*/*softgoal* or *resource*.



I-STAR (i*)	GOALS
Actor Boundary	Lines Vis./Int.
Actor Agent Role Position	Actor
	BP
Belief	*Belief*
Goal	UT
Softgoal Task	U.Intention
	U.Interaction
Actor Boundary	IX
	AS
	IC
	IO
	BR
	UISR
	BDSR
Resource	DE

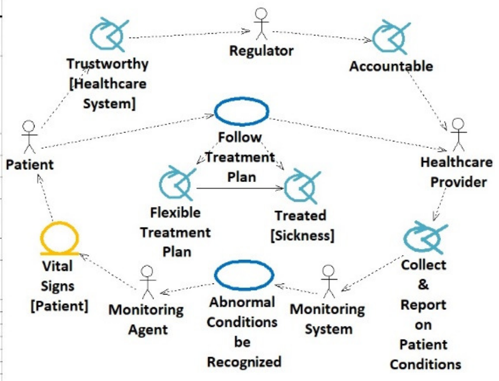


Figure 79. I* and GOALS equivalent Representation.

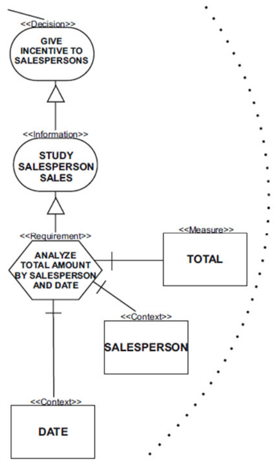
In GOALS, natively, the shared concept is the Interaction Space, in which *actors* perform each his own User Task, subject to the same rules and sharing the same *data*. I* must also be considered as a *business process* model, since it defines an ongoing permanent situation, until a *goal* is achieved or one of its components is not available anymore.

GOALS further structures Business Rules when compared with I*, and I* further introduces beliefs (only available in the GOALS *ideology*) and positions as a further specification of the *actor* social context. GOALS may conceptually be used with I* in order to further compose the Enterprise Structure with a social perspective.

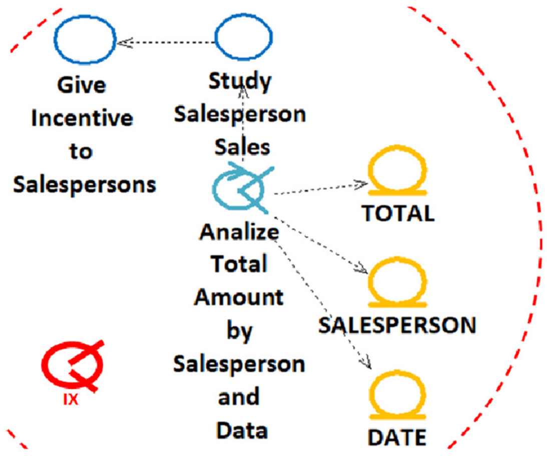
MAZÓN

MAZÓN [Mazón et al., 2007], presents a method that uses the I* framework to model the *requirements* and the Database of a *data* warehouse. It uses the concepts of (from more abstract to more specific) strategic *goal*, decision *goal* and information *goal* in order to derive the fact tables, the bases and dimensions of a *data* warehouse. The model is developed by means of the decomposition of the hierarchy of *goals* and the identification of the *tasks* that support the achievement of each *goal*. Based on the *tasks*, the information *requirements* are identified, and the fact tables and their dimensions (or base) are identified and related.

Figure 80 presents the computation independent model of the method and the equivalent representation using the GOALS language. *Goals* and *tasks* are expressed by User Intentions, and the information and *resources* are represented using Data Entities (DE).



MÁZON	GOALS
	Lines Vis./Int.
Actor	Actor
	BP
	Belief
Goal	UT
Goal / Task	U.Intention
	U.Interaction
Actor Boundary	IX
	AS
	IC
	IO
	BR
	UISR
	BDSR
Resource	DE



Give Incentive to Salespersons
Study Salesperson Sales
Analize Total Amount by Salesperson and Data
TOTAL
SALESPERSON
DATE
IX

Figure 80. MAZÓN and GOALS equivalent Representation.

The structure of concepts is different, but the logic of relation is the same, as it relates the *data* that should be used to satisfy the usage of the system which will support achieving the *goal*. This relation is not sufficient in order to model the User Interface, but the establishment of this relation compares to the relation which is established in the GOALS Service Design (Step 1) between Business Processes and DEs i.e. the relation is not fully structured in terms of user actions, but it is possible to perceive that it will be used for the purpose of the human activity.

6.2.6. Software Engineering

The RATIONAL UNIFIED PROCESS (RUP) and the UNIFIED MODELING LANGUAGE (UML) are the de facto standards as Software Development Process (SDP), for *requirements* elicitation and for software architecture elaboration. Many methods apply the combined RUP and UML Object-Oriented (OO) analysis and design techniques in order to develop the internal structure of the Information System. The RUP and the derived methods are analyzed concerning *use cases* elicitation and the internal structure of the Information System elaboration.

RATIONAL UNIFIED PROCESS (RUP)

The RUP Software Development Process (SDP) is composed of the phases of "Inception", "Elaboration", "Construction" and "Transition", which gradually build the Information System based on the "Business Modeling", "Requirements", "Analysis & Design", "Implementation", "Test" and "Deployment" process workflows.

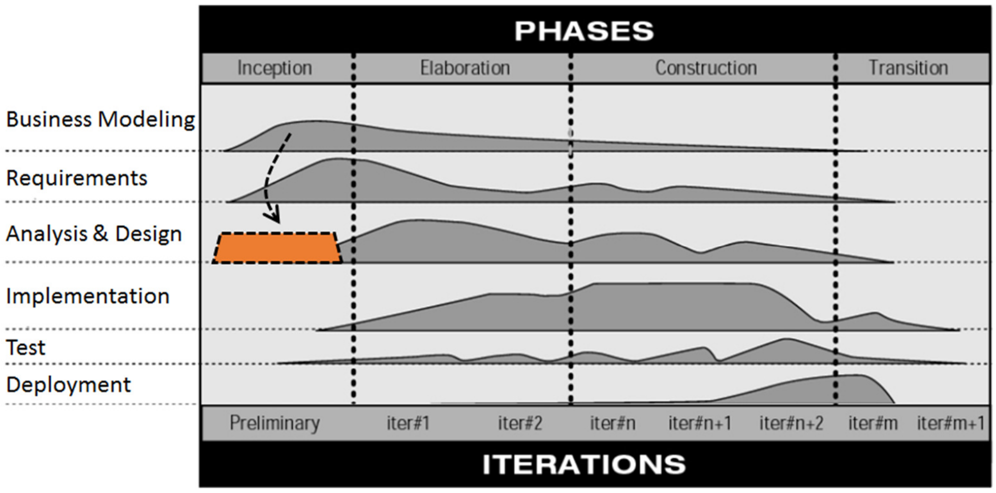
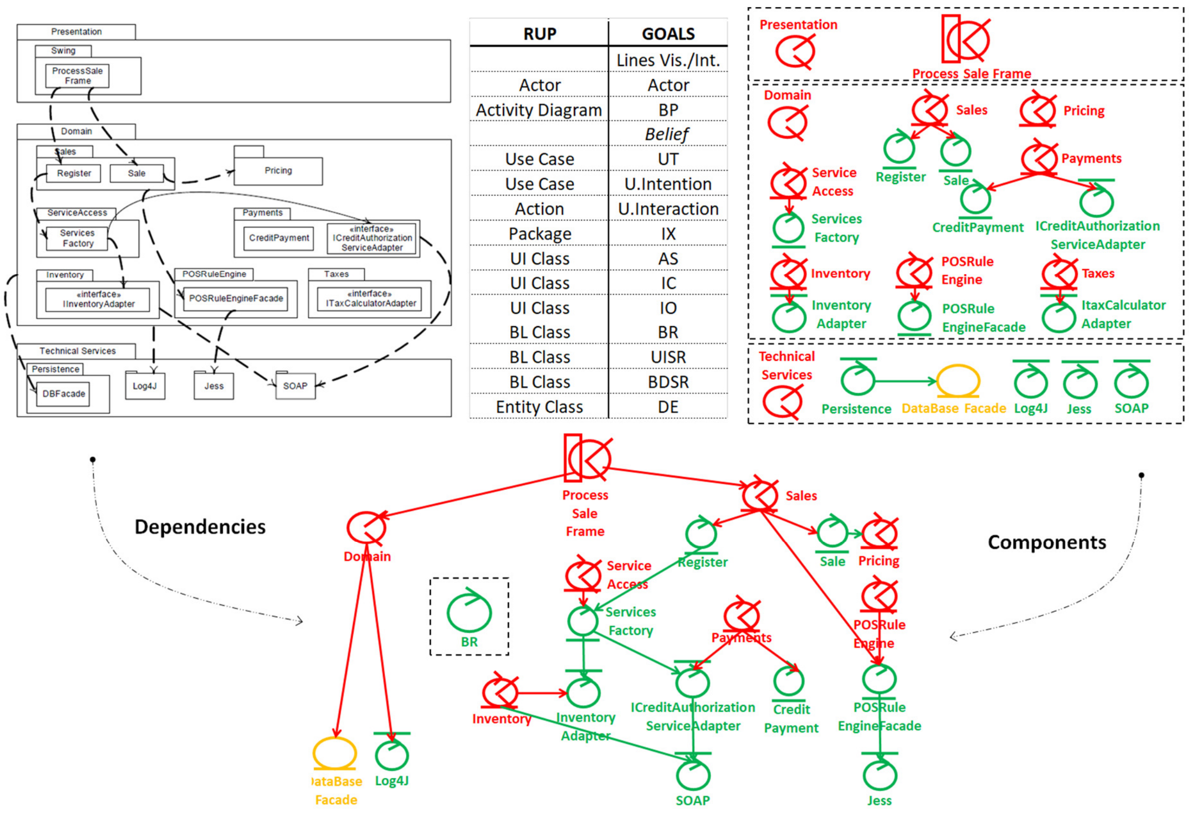


Figure 81. GOALS Change relatively to the RUP SDP.

The RUP and GOALS SDPs may can be compared in the perspectives of the "Business Modeling", "Requirements" and "Analysis & Design", however with changes, as the "Business Modeling" workflow may be considered as already included in the "Analysis & Design" by carrying on iteratively the Analysis (business model) and Design (software model) phases. Consequently, as *requirements* are already identified, the "Requirements" workflow may be considered for non-functional requirements only. The change is highlighted in Figure 81 (orange dashed line trapezoid), and represents an improvement to the SDP with one less workflow ("Business Modeling"), and a "thinner" "Requirements" workflow. This change brings clarification and efficiency to the SDP, since the three phases are reduced to two and performed in a single logical sequence (software design after business analysis), applying techniques that induce a more straight-lined and well-defined relation of business and software.

Independently of SDP, the structure of *requirements* is the focus of our comparison, as it drives the system design. RUP focuses its target in the elaboration of the internal structure of the Information System based on OO *business concepts*, in order to provide support for the needed functionality as identified in *use cases*. This may be considered as more architecture-centric than human-centric approach, focusing more on the software structure and functionality than on how the user completes a *task*, which in contexts where UX and performance matters, makes the difference. Therefore, the most commonly observed problem of the application of RUP is related to the User Interface design and usability [Nunes, 2009], as the elicited *use cases* do not have a suitable well-defined level of granularity when considered in the business environment. The *Use case* may represent a full set of *tasks*, a single *task*, or a "click" *task*, and may be related by "inclusion" and "extension" sub-dividing responsibility, but its business representation continues limited as they do not become "meaningful" in a business perspective, as they do not represent a full *task*, not allowing the specification of a uniform logic of communication between *actors* . Without a business-centric organization, the architectural organization is driven by system functionality, creating levels of dependency that may become unmanageable.

Grouping functions by functionality raises a problem of modifiability [Bachmann et al., 2007], a situation that similarly also applies to the Normalized Systems [Bruyn & Mannaert, 2012], being therefore independent from the method, as the coupling between packages (or modules) is a problem of organization, more than of functionality or system design, which in the limit only exists because the deployment demands it.



RUP	GOALS
	Lines Vis./Int.
Actor	Actor
Activity Diagram	BP
	Belief
Use Case	UT
Use Case	U.Intention
Action	U.Interaction
Package	IX
UI Class	AS
UI Class	IC
UI Class	IO
BL Class	BR
BL Class	UISR
BL Class	BDSR
Entity Class	DE

Figure 82. RUP and GOALS Architectural Organization.

In the example of architecture organization of LARMAN [Larman, 2001]) presented in Figure 82, functionality is organized in the "Presentation", "Domain" and "Persistence" packages. By means of mapping: the *package* to the Interaction Space (IX, gathering *data* and *regulations*); the *internal packages* to Aggregation Spaces (AS), Interaction Components (IC, gathering functionality from *business concepts* e.g. "Sales"), and Data Entities (DE) (targeting MVC); the "«*interface*»" classes to UISRs; and the remaining functions to DBSRs, we derive the same architecture in GOALS, where the organization is now business-centric, and based on *business* model concepts (the "Domain" IX and the "Database Façade" DE), proving insight on what would be equivalent RUP and GOALS architectural structures.

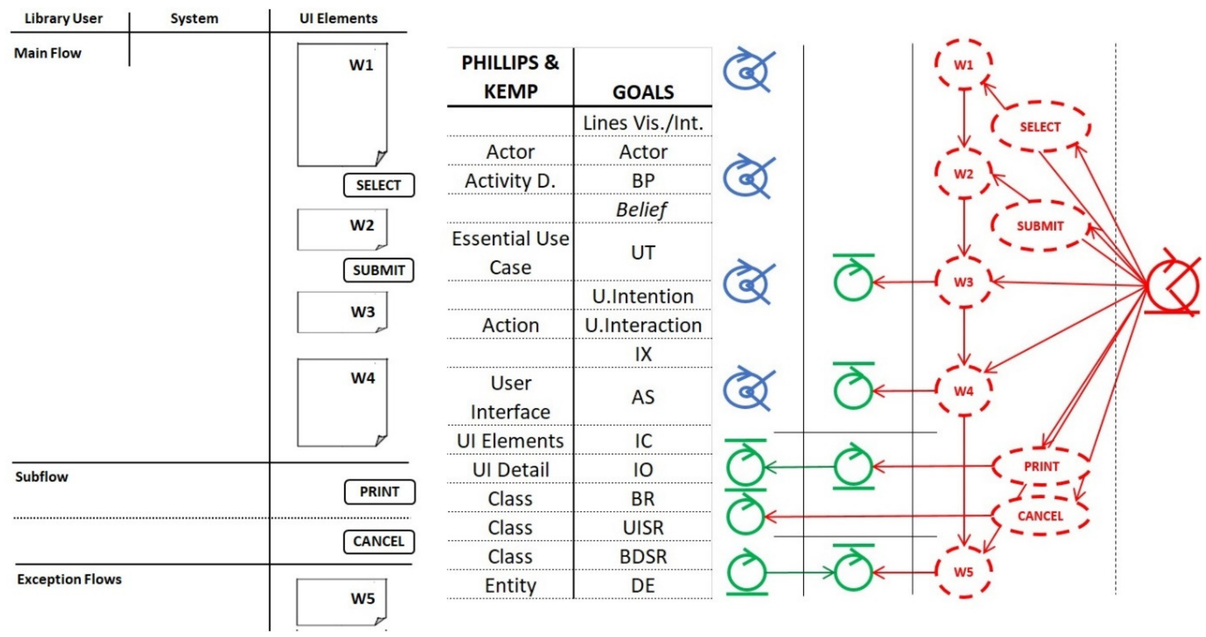
GOALS is independent from implementation, and therefore, each implementation concept may be supported by a distinct technology, whether a software-specific (AS, IC, IO, UISR or BDSR) of hybrid (IX, BR or DE) component. Once components are grouped around *business concepts* they become "business-centric" i.e. the organization is based on Business Processes (BP) (which concerns are usually distinct), providing an affinity logic which is refined by the sub-division of the BP in UTs performed in IXs (managing the same business *concepts*). This promotes a high level of coupling to each IX of the BP, and a lower level of coupling between BPs. The business-centric organization also allows an easier identification of component implementation priority, facilitating work distribution among the software development team.

The (undefined) level of granularity of the RUP *use cases* also contributes to generate the recurrent need for refactoring, since when new *tasks* which are similar to already existing *tasks* are introduced by new *business processes*, there is the need to use the already implemented functions in order to avoid repeating code. And when new parameters are needed, the need to refactor at least a single package will probably exist, in which situation the same need may also be propagated to distinct packages that support other *business processes*. This is an issue that GOALS contributes to diminish by means of the elaboration of the Software Architecture in terms of UTs of the same Interaction Space (IX), promoting the reuse of the User Interface components (AS, IC or IO) within that logical space, also preparing the Business Logic layer functions to support more than one invoking component, and therefore inducing the generalization of components from conception (design-time), contributing to avoid refactoring. Another important cause that may lead to refactoring efforts, is that the functionality enclosed in packages (in order to be used by functions of other packages) that implement business *regulations*, may be programmed in multiple functions, that provide and receive *data* from the User Interface, and not necessarily isolated in a single function, raising (business *regulations*) modifiability issues. By means of isolating *regulations* in Business Rules (BR), GOALS clearly separates the business and the software (logic) concerns, "dodging" the need to organize packages, promoting the BR as the more important element of the Business Logic, and specializing the remaining SRs solely in the management and communication of *data*, so that they may be programmed separately, as business *regulations* are left out of the scope of their concerns. For example, the architecture presented using the GOALS language in Figure 82 is based on the dependencies presented in the LARMAN architecture, from where it is possible to identify that according to the Enterprise Structure model, the missing element is the Business Rule (BR) as the mediating component between IX and DE.

The RUP Domain Model is left unchanged along the GOALS method, since it is from it, and by maintaining the elaboration of this model along the SDP, that the affinity relations valid for the elaboration of the RUP packages, are also used for the specification (and reuse) of the User Interface (AS, IC, and IO), and the supporting Business Logic, which only after are related to the Domain Model classes (Data Entities, DE) and attributes (DE Fields). Summarizing, when compared to RUP, GOALS maintains the same Domain, but uses *essential use cases*, and from them, builds first the User Interface, and only after an equivalent Business Logic.

PHILLIPS & KEMP

The PHILLIPS & KEMP method [Phillips & Kemp, 2002], includes a "Main Flow", "Subflows" and "Exception Flows", from which it derives components of the User Interface being conceived. Figure 83 presents one example (adapted) of architecture derived from the application of the method. The identified problem is the granularity of reuse at the level of the user interaction, reflecting the (usually) lower level of abstraction of the *use case* when it is not essential (considering the complete *task* within the BP flow), impeding that the level of reuse may be identified at a higher-level "tool", "User Intention", or "set-of-interactions". For example, the "W1" User Interface (book *data*: "Title", etc.), may be needed in other parts of the system. And since there is the probability that this system is not implemented independently of the remaining *use case* information, this part may probably be implemented at first as part of the "Request Book" *use case*, and not as an isolated and reusable component, augmenting the probabilities of architectural reorganization especially when new semantically close *use cases* are introduced to the system.



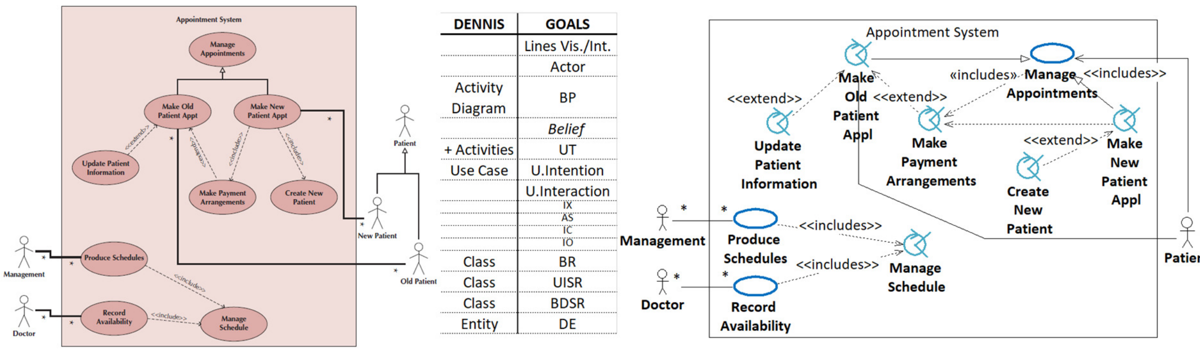
PHILLIPS & KEMP	GOALS
	Lines Vis./Int.
Actor	Actor
Activity D.	BP
	Belief
Essential Use Case	UT
	U.Intention
Action	U.Interaction
	IX
User Interface	AS
UI Elements	IC
UI Detail	IO
Class	BR
Class	UISR
Class	BDSR
Entity	DE

Figure 83. RUP PHILLIPS & KEMP and GOALS User Interface Architecture.

The problem may be worst when *use cases* may be used to model the *business process* of the enterprise using Activity Diagrams, since there is the tendency to specify more than one *use case* per *actor* in the course of the *business process*, as a complete user *task* will usually involve *use cases* which are part of other complete *tasks*. In GOALS the complete *tasks* are *use cases* (User Tasks) and by granularity level, the *use cases* usually identified with RUP, are at the level of the User Intention, settling a higher level of abstraction of the User Task (UT) relatively to the *essential use cases* applied by the PHILLIPS & KEMP method. In this situation the *use cases* will probably not be exclusive in terms of the used *data* and components, and for that reason, if the reuse and structuring of the components is not carried out in design time, it will be almost impossible to ensure that the architectural control and total reuse of components is maintained without a refactoring overhead effort.

DENNIS

The DENNIS method [Dennis et al., 2012], is an example where the same situation of *use case* "Indefiniteness" happens. In this case, after being derived from business processes modeled using Activity Diagrams, *use cases* were iteratively rearranged, yet losing all possibilities of maintaining traceability of the business and software models.



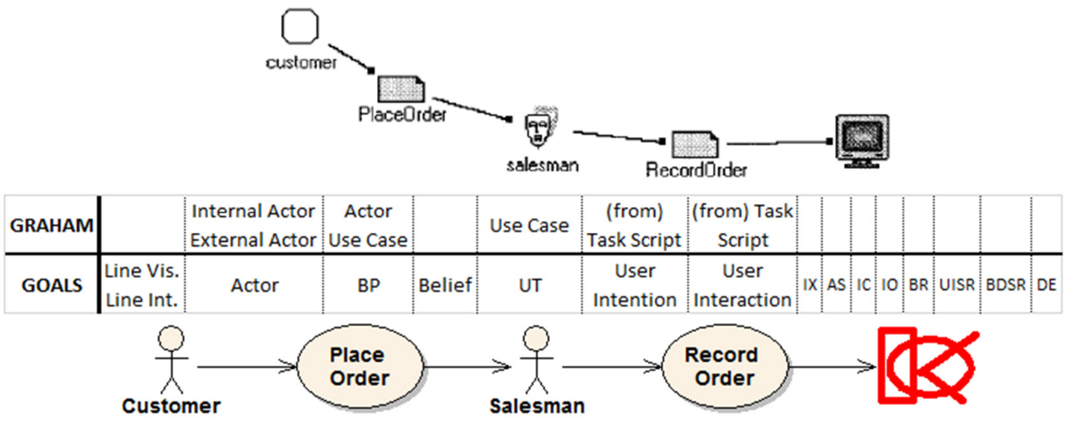
DENNIS	GOALS
	Lines Vis./Int.
	Actor
Activity Diagram	BP
	Belief
+ Activities	UT
Use Case	U.Intention
	U.Interaction
	IX
	AS
	IC
	IO
Class	BR
Class	UISR
Class	BDSR
Entity	DE

Figure 84. DENNIS and GOALS Relation of Use Case Concepts.

Figure 84 presents an example of a *use cases* model which was derived using the DENNIS method. Each "extended" and "included" *use case* are at the granularity level of GOALS User Intentions, as they are part of a bigger *task*. Each of the User Intentions will have a correspondence with an Interaction Component (IC), that will have greater probabilities of being identified for reuse, since it has a higher level of granularity than the interactions predicted in the *use cases* specifications.

GRAHAM

Curiously, the method from GRAHAM [Graham, 1996], in 1986, already provided an interesting approach to surpass the granularity problem, and provide a similar form of communication between *actors* to *goals*, by means of using a model of *business process* which is also similar to I* [Yu, 1995] in order to withdraw *use cases*. As the *use cases* are seen as the linking artefact between *actors*, the indefiniteness and granularity problem is eliminated. Figure 85 exemplifies the situation, where the "Customer" places an order, and the "Salesman" executes it over the system, in a cleaner (easier to read) and more consistent (less probability, more resistant, to change) representation of *business processes* using *use cases*. Thus, the level of granularity of the GRAHAM *use case* is precisely the same than in GOALS.



GRAHAM		Internal Actor External Actor	Actor Use Case		Use Case	(from) Task Script	(from) Task Script								
GOALS	Line Vis. Line Int.	Actor	BP	Belief	UT	User Intention	User Interaction	IX	AS	IC	IO	BR	UISR	BDSR	DE

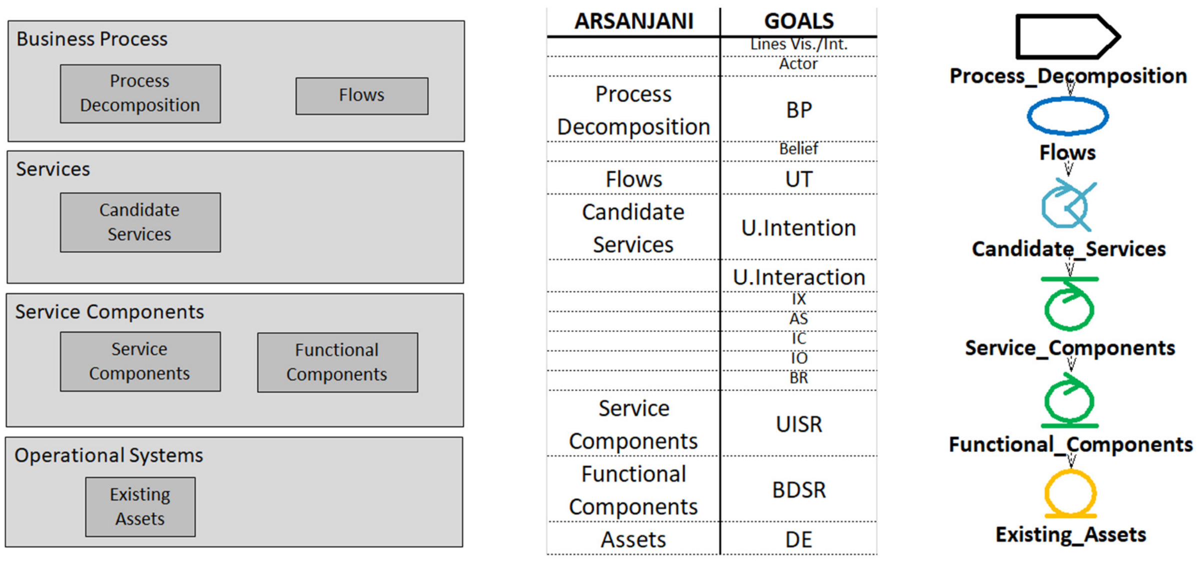
Figure 85. GRAHAM and GOALS Similarities to I*.

6.2.7. Service Oriented Architecture

Service oriented architectures (SOA) provide services of information that are usually used by remote enterprise' systems in order to enable commercial relations automatically, eventually without other human intervention than the customer. What is important is the structure of information transferred from one enterprise to the other, and the reference to an ongoing business *transaction*. For this purpose, in the terms of the human activity perspective, the user intentions have a level of granularity that suits the structuring of the information to be transferred, which will also be subject to commands i.e. functions that receive and records *data*.

ARSANJANI

The ARSANJANI method includes an extensive SOA framework that organizes the *business processes* and formulates the services that must be implemented in order to supply external enterprise connectivity needs. Functionality is organized in packages, but since it is elicited based on a business perspective, its identification and organization becomes independent from deployment, separating the operative system needs from the semantic model.



ARSANJANI	GOALS
	Lines Vis./Int.
	Actor
Process Decomposition	BP
	Belief
Flows	UT
Candidate Services	U.Intention
	U.Interaction
	IX
	AS
	IC
	IO
	BR
Service Components	UISR
Functional Components	BDSR
Assets	DE

Figure 86. ARSANJANI and GOALS SOA Representation.

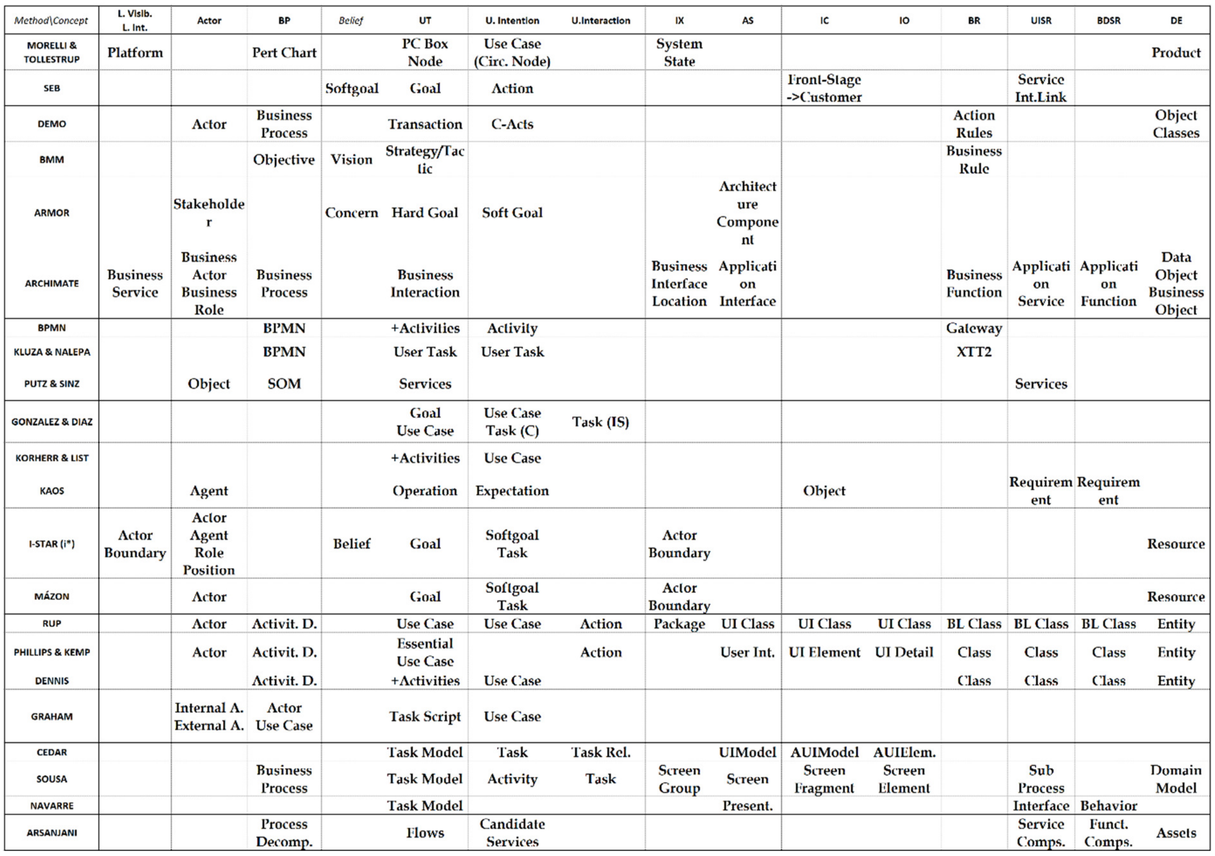
Figure 86 presents the SOA framework [Arsanjani et al., 2008] and the meta-model of its conceptual structure as it may be designed using GOALS, where the User Intentions “Candidate Services” are directly related to the User Interface System Responsibilities establishing a SOA structuring logic.

6.3. Validation Conclusions

This chapter aimed at validating the GOALS language and method for the purpose of Information System analysis and design. We provided a cross-consistency validation, and also a perspective of the power of expression of the language, in terms of scope and detail, by means of modeling other methods diagrams using GOALS, highlighting the differences. This allows understanding its ease of use, since by means of seeing the same information modeled by two languages, the modeler may understand what is the notation and method that suits his needs, since as long as the used concepts are compatible, any language may be used to complete the enterprise architecture.

The more important and structuring comparisons are relative to DEMO and RUP methods. On one hand, GOALS is structured on DEMO for the specification of a software model that supports any possible combination of *business processes* based on *transactions*. This is a validation which “entitles” the GOALS business model to be included as the “back-bone” structure of a Software Architecture composed by means of the detail of the human activity. On the other hand, relatively to RUP, the inclusion of the business “back-bone” in the software model allows the specification of the Information System in a top-down process benefiting the conception of the User Interface and the isolation of business *regulations* in the Business Logic, promoting the system modularization. The specification of the Database remains compatible, acknowledging however in advance which MVC components will use which Tables and Fields.

Considering current SE practice, GOALS Software Architecture may be considered as business-centric, suggesting changes to the organization of packages, since the Interaction Space (IX) allows establishing a relation to all the software components of the User Tasks (UT) which are performed on it, including the User Interface, Business Logic and Database in a structure of usage, providing new reference terms for packages creation, centered in business functionality (the *business processes*) instead of software functionality (the *use cases*).



Method\Concept	L. Visib. L. Int.	Actor	BP	*Belief*	UT	U. Intention	U.Interaction	IX	AS	IC	IO	BR	UISR	BDSR	DE
MORELLI & TOLLESTRUP	Platform		Pert Chart		PC Box Node	Use Case (Circ. Node)		System State							Product
SEB				Softgoal	Goal	Action				Front-Stage ->Customer			Service Int.Link		
DEMO		Actor	Business Process		Transaction	C-Acts						Action Rules			Object Classes
BMM			Objective	Vision	Strategy/Tac tic							Business Rule			
ARMOR		Stakeholde r		Concern	Hard Goal	Soft Goal			Architect ure Compone nt						
ARCHIMATE	Business Service	Business Actor Business Role	Business Process		Business Interaction			Business Interface Location	Applicati on Interface			Business Function	Applicati on Service	Applicati on Function	Data Object Business Object
BPMN			BPMN		+Activities	Activity						Gateway			
KLUZA & NALEPA			BPMN		User Task	User Task						XTT2			
PUTZ & SINZ		Object	SOM		Services								Services		
GONZALEZ & DIAZ					Goal Use Case	Use Case Task (C)	Task (IS)								
KORHERR & LIST					+Activities	Use Case									
KAOS		Agent			Operation	Expectation				Object			Requirem ent	Requirem ent	
I-STAR (i*)	Actor Boundary	Actor Agent Role Position		Belief	Goal	Softgoal Task		Actor Boundary							Resource
MÁZON		Actor			Goal	Softgoal Task		Actor Boundary							Resource
RUP		Actor	Activit. D.		Use Case	Use Case	Action	Package	UI Class	UI Class	UI Class	BL Class	BL Class	BL Class	Entity
PHILLIPS & KEMP		Actor	Activit. D.		Essential Use Case		Action		User Int.	UI Element	UI Detail	Class	Class	Class	Entity
DENNIS			Activit. D.		+Activities	Use Case						Class	Class	Class	Entity
GRAHAM		Internal A. External A.	Actor Use Case		Task Script	Use Case									
CEDAR					Task Model	Task	Task Rel.		UIModel	AUIModel	AUIElem.				
SOUSA			Business Process		Task Model	Activity	Task	Screen Group	Screen	Screen Fragment	Screen Element		Sub Process		Domain Model
NAVARRE					Task Model				Present.				Interface	Behavior	
ARSANJANI			Process Decomp.		Flows	Candidate Services							Service Comps.	Funct. Comps.	Assets

Figure 87. GOALS and Related Work Concepts Comparison.

Figure 87 presents the mapping of all methods compared with the GOALS model, where it is possible to identify that business-oriented methods (namely ARCHIMATE) cover a great part, considering the lines of interaction and visibility, also structuring the concept of *belief* (BMM and ARMOR) as a relevant social aspect of enterprise and system engineering. The software-oriented RUP-based methods cover the GOALS model in a good extend, from which they derive the needed software architectures, however not ensuring a clear line of traceability between the business representation and the *use cases* identification. Moreover, it is also important to observe how methods may be combined to extend their solution to the areas of the problem which they do not address, as long as the used concepts remain compatible.

GOALS is not compared directly to agile methods, namely XP, SCRUM and SAFE, which processes that guide the software project, in which the precision and traceability of the software architecture is not a relevant. The Agile Manifesto software development values "Individuals and interactions"; "Working Software"; "Customer Collaboration"; and "Responding to Change". GOALS contributes to favor those values by means organizing what the manifesto "leaves behind", by means of organizing in 8 Steps the "Processes and (their) Tools" in terms of Business Processes (BP), User Tasks (UT) and Interaction Components (tools of a User Interface), whereas development tools are out of the scope of current study; in order to produce the minimal "Comprehensive Documentation" in 4 business (Steps 1 to 4) and 5 software models (5 to 8), including and Interaction Model, the User Interface design and the Information System specification in an MVC pattern; in order to facilitate "Contract Negotiation", based on less and more easily readable models where the physical and logical spaces are also modeled, providing the space to apply user-centered techniques that favor agile development; which facilitates "Following a Plan" driven by the business model and the customer needs.

7. Conclusions

This thesis presented the GOALS language and method, which may be used to model the enterprise and its software Information System, ensuring business and software traceability. The method may be used to architect the business according to the needs of the users, customers and other parties, which are related to the enterprise activity. It details the activity in terms of user interactions and software components that must be programmed to support it, and provide adequate User Interface behavior, Business Logic processing, and a Database structured on enterprise *business concepts*.

GOALS applies the method based on an Enterprise Development Process (EDP) oriented by a strategy of evolution of the logical spaces of the enterprise, both in their physical and technological perspectives: the interactive spaces (of human and Information System: User Task, Interaction Space, Aggregation Space); interactive tools (User Intention, Interaction Component) and interactive objects (User Interactions, Interaction Objects, System Responsibilities, and Fields of Data Entities:). This logic provides perspective on the environment of the enterprise and the user context in function of the *business processes goals*. The *business processes* and *actor* relation also provides a service social perspective of the organization of the enterprise concerning the initiators and participants, and their relation with the customer, and the results that they use and produce in terms of *business concepts*.

GOALS finds the “space” to fill the “gap” between enterprise and software engineering in the modeling of the spaces of the enterprise, where the human-computer interaction techniques are applied in order to produce User Interface architectures, their behavior and the relation to the remaining internal structure of the information system. The used techniques are based on *task* and interaction models, which have a relation with the higher and lower level functions of the Business Logic of the Information System. Additionally, the higher level functions may be used for the specification of service clusters of functions, which may be used for SOA in order to specify web services clusters, while the lower level functions provide the nuclear information and commands of the system. In the same way that SOA establishes a relation between the business and the Business Logic of the system, GOALS exemplifies that the alignment between enterprise and software engineering processes can also be achieved in order to provide support for the cooperation of EE and SE towards the objective of implementing the Information System using a well-defined and mainly more time-efficient enterprise and software development process, challenging the idea that there is an absolute need to define separate ontologies for enterprise and information system as defined in the EE manifesto [Dietz, 2014].

Independently of the covered ZF cells filled by GOALS and by the related methods (as presented in Chapter 2. State of The Art), we believe that the main contribution of our language is the consistency of the relation between business and software which is achieved by means of the representation of the *business process* design using *essential use cases*, promoting a stable logic of modeling. With it, “Indefiniteness” at the *use case* level “disappears”, as *essential use cases* becomes a complete *business process task*, establishing its user intentions at the same level of granularity of the “traditional” *use case* (as in most cases more than one is needed to complete a *business process* task), and finishing at the interaction level. This level of granularity of specification allows the application of agile development techniques that may be used to improve a single *use case*, a User Intention, a User Interaction, or the *data* contained in each field, and its form (relative to semiotics): *syntagmatic*, *paradigmatic* or *syntactic* information.

The contribution of our proposal is mostly related to the Software Engineering (SE) domain, from which our method is originated, to which we add an enterprise context that facilitates understanding the *requirements* that the system must conform with, namely the used spaces, the business *regulations* that apply, and the information that it must hold. It is by means of separating the design from implementation, and by elaborating the affinity of the business components first, and only after, of the semantically related software-specific components, that we "dodge" the need to create packages based on functionality, shifting to a business-centric organization, that promotes cohesion with identified Interaction Spaces, also diminishing it between distinct BPs, maintaining traceability by means of the relation of the business and software architectures. By means of representing the business structure in the software structure, we contribute to diminish the problem of tracing models from the *requirements* throughout the *design* phase, which is also a recurrent in Software Product Lines.

The GOALS language may be used by existing development processes (XP, SCRUM, SAFE and RUP) depending on the desired level of specification, targeting the elaboration of architectural "spikes" that may be implemented in increments, matching agile software architecture *requirements* [Grundy, 2013]. Moreover, the GOALS language may also be used to reflect enterprise architectures modeled using the DEMO method as a form of translation that facilitates software development. This is carried out by means of relating DEMO interrelated Transactions, their Action Rules and Objects Classes to the User Tasks, Business Rules and Data Entities of the GOALS language, and further complementing it with the Interaction Spaces where the human activity occurs.

Besides the presented contents, GOALS induces a new way of thinking about software and the modeling of the enterprise, as it tries to embrace all the complexity involved in an enterprise improvement, whether business or software, and preferably both. The business complexity is expressed from the design of the *service* and the *business process*, to the user *task*, intention and interaction. And each has a direct or patterned relation with the parts of the supporting Software Architecture. Attention is also paid to the strategy of the EDP, privileging the identification of working spaces, and its transformation in business processes and working software.

This "straight line" between business and software modeling can only be achieved because the modeling of the business is already the modeling of the software, as the business model is "natively" integrated in the software model. The implementation-independent model allows the "reflection" of the problem in the solution and leaves the implementation responsibility to software managers, knowing in advance, that all the Software Architecture components can be implemented in a relation of one (component) to one (e.g. file or table) implementation object, depending on the chosen implementation technology.

The presented work also benefits from the contribution of The Timeless Way of Building [Alexander, 1979], which induces the elaboration of building artefacts according to the needs, being these requirements manifestly expressed, or observed by the architect. The models which are produced are subject to suitable established patterns, which should be replicated in order to perfect the process of building. In the same way, by means of observing what is really important in each person daily life, including the user, manager, software practitioner and system needs, we established a solution based on accepted patterns that provides support for recurrently observable enterprise business and software development problems.

7.1. Future Work

In a stable production environment, which uses software patterns in order to augment the software production performance, it is possible to understand what may be the reference software architecture for a given business development problem. Then, by means of direct comparison, it may be possible to understand that are the design decisions that lead to cheaper or more expensive software development and provide and increased or decreased Return of the Investment (ROI). Thus, by means of analyzing the business it is possible to understand the benefits and expected return, and by means of the modeling of the software it will be possible to estimate the cost of development, opening a space and an ultimate software challenge, which is the "art" of predicting the software development effort with accuracy and generating usable software from business and software models.

7.1.1. Platform Specific Model for Information System Generation

We believe that by means of the establishment of a Platform Specific Model (PSM) it is possible to generate the complete structure of an enterprise Information System Software Architecture. For that purpose, the level of detail and formalization of the GOALS method needs to be increased with the customization concerning the historical presentation of *data*, the level of User Interface feedback, and the full specification of the semiotic logic.

7.1.2. Software Product Lines

GOALS is oriented to the development of a single system, yet, we believe that the proposed architectural solution may contribute to Software Product Lines (SPL) by means of facilitating the identification of the valid versions of the implemented objects. By means of implementing the versioning of the produced architectures (including their components), we may help the objective of conciliating distinct architectures (SPLs), identify the compliant components and transporting those parts to new architectures, where the *requirements* must be updated.

7.1.3. Software Effort Estimation

Based on the application of the GOALS method in a software development environment, we expect to be able to better control the estimation and execution of software development in order to further enhance the Use Case Points formula [Alves et al., 2013; Karner, 1993]. We believe that if estimation may be based on the size of the *business process* in terms of *use cases*, the number of *regulations* and managed *data* entities, and also the User Interface design option, then it may be possible to contribute to increase software project accuracy.

7.1.4. Goals Modeling Tool

The following stage of the our future work aims the development of a GOALS language modeling tool that may apply the process and method using the language with architectural control, and the possibility of managing the software project, controlling estimation and implementation, and generation and testing of the final Information System.

References

Akiki, P. (2013). *Engineering adaptive user interfaces for enterprise applications. PhD Thesis*. Open University.

Alexander, C. (1979). The Timeless Way of Building. *New York Oxford University Press*. doi:10.1080/00918360802623131

Alves, R., Valente, P., & Nunes, N. J. (2013). Improving Software Effort Estimation with Human-centric Models: A Comparison of UCP and iUCP Accuracy. *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 287–296. doi:10.1145/2480296.2480300

Alves, R., Valente, P., & Nunes, N. J. (2014). The State of User Experience Evaluation Practice. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational* (pp. 93–102). New York, NY, USA: ACM. doi:10.1145/2639189.2641208

Armstrong, C., Baker, J. D., Band, I., Courtney, S., Jonkers, H., Muchandi, V., & Owen, M. (2013). *Using the Archimate Language with UML.*

Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., & Holley, K. (2008). SOMA: a method for developing service-oriented solutions. *IBM Systems Journal*, *47*(3), 377–396. doi:10.1147/sj.473.0377

Aveiro, D., Pergl, R., & Valenta, M. (Eds.). (2015). Advances in Enterprise Engineering IX. In *Proceedings of the 5th Enterprise Engineering Working Conference (EEWC)*. Prague, Czech Republic: Springer International Publishing. doi:10.1007/978-3-319-19297-0

Bachmann, F., Bass, L., & Nord, R. (2007). Modifiability Tactics. *Software Engineering Institute*, *Technical*(September). www.sei.cmu.edu/publications/pubweb.html

Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70–77. doi:10.1109/2.796139

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). Manifesto for Agile Software Development. http://agilemanifesto.org/. Accessed 12 December 2017

Becker, J., Knackstedt, R., Matzner, M., & Pöppelbuß, J. (2008). Leveraging Product-Service Systems by Implementing Service-oriented Architecture. In *International RESER Conference 2008* (pp. 1–17).

Bell, M. (2008). *Service-oriented modeling (SOA): service analysis, design, and architecture*. Wiley. ISBN: 978-0-470-14111-3

Berg, K., Bishop, J., & Muthig, D. (2005). Tracing software product line variability: from problem to solution space. *Proceedings of SAICSIT 2005*, 182–191. ISBN: 1-59593-258-5

Biddle, R., Noble, J., & Tempero, E. (2002). Essential Use Cases and Responsibility in Object-oriented Development. In *Proc. of 25th ACSC '02* (pp. 7–16). Darlinghurst, Australia. ISBN: 0-909925-82-8

Booch, B. G., Rumbaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language User Guide* (2nd ed.). Addison-Wesley. ISBN: 0321267974

Bowen, J., & Dittmar, A. (2016). A semi-formal framework for describing interaction design spaces. *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '16*, 229–238. doi:10.1145/2933242.2933247

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004). Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, *8*(3), 203–236. doi:10.1023/B:AGNT.0000018806.20944.ef

Breu, R., Agreiter, B., Farwick, M., Felderer, M., Hafner, M., & Innerhofer-oberperfler, F. (2011). Living Models – Ten Principles for Change-Driven Software Engineering. *International Journal of Software and Informatics*, *5*(231101), 267–290.

Bruyn, P. De, & Mannaert, H. (2012). Towards applying normalized systems concepts to modularity and the systems engineering process. *ICONS 2012 : The Seventh International Conference on Systems*, (c), 59–66. ISBN: 9781612081847

Cardona, V., & Duarte, H. (2013). Approach for the Model Driven Development of Business Processes Lines Based on Service Oriented Architectures. *American International Journal of Contemporary Research*, *3*(7), 78–87.

Chelimsky, D., Astels, D., Dennis, Z., Hellesøy, A., Helmkamp, B., & North, D. (2010). *The RSpec Book - Behaviour-Driven Development with RSpec, Cucumber, and Friends*. The Pragmatic Bookshelf. ISBN: 978-1-93435-637-1

Constantine, L. L. (2006). Activity Modeling. *ReVision*, *35*, 27–51. doi:10.1007/978-1-84800-907-3_3

Constantine, L. L., & Lockwood, L. a. . (2001). Structure and style in use cases for user interface design. *Object Modeling and User Interface Design.*, *1*(July 2015), 245–280. doi:10.1.1.94.8255

Constantine, L., Windl, H., Noble, J., & Lockwood, L. (2003). *From Abstraction to Realization: Canonical Abstract Prototypes for User Interface Design REVISED. Distribution* (Vol. 1). doi:10.1287/opre.1080.0628

Costa, D., Nobrega, L., & Nunes, N. J. (2007). An MDA approach for generating web interfaces with UML ConcurTaskTrees and canonical abstract prototypes. *Lecture Notes in Computer Science*, *4385*, 137–152. doi:10.1007/978-3-540-70816-2_11

Dai, L., & Cooper, K. (2007). Using FDAF to bridge the gap between enterprise and software architectures for security. *Science of Computer Programming*, *66*(1), 87–102. doi:10.1016/j.scico.2006.10.010

Damjanovic, V., Gasevic, D., & Devedzic, V. (2005). Semiotics for ontologies and knowledge representation. *CEUR Workshop Proceedings*, *130*, 1–4.

de Vries, M., van der Merwe, A., & Gerber, A. (2017). Extending the enterprise evolution contextualisation model. *Enterprise Information Systems*, *11*(6), 787–827. doi:10.1080/17517575.2015.1090629

Decreus, K., & Poels, G. (2011). A Goal-Oriented Requirements Engineering Method for Business Processes. *Information Systems Evolution SE - 3*, *72*, 29–43. doi:10.1007/978-3-642-17722-4_3

Delgado, A., Ruiz, F., Guzman, I. G.-R. De, & Piattini, M. (2010). A Model-driven and Service-oriented framework for the business process improvement. *Journal of Systems Integration*, *1*(3), 45–55. http://www.si-journal.org/index.php/JSI/article/view/55

Dennis, A., Wixom, B. H., & Tegarden, D. (2012). *Systems Analysis and Design with UML*. Wiley. ISBN: 978-1-118-03742-3

Development, C. (2010). CMMI for Development, Version 1.3, (November).

Dietz, J. L. G. (2006). *Enterprise ontology*. New York: Springer. doi:10.1007/3-540-33149-2

Dietz, J. L. G. (2014). Enterprise engineering - The manifesto. *Lecture Notes in Business Information Processing*, *174 LNBIP*(January), 1–2. doi:10.1007/978-3-319-06505-2

Dubberly, H., & Evenson, S. (2008). The experience cycle. *Interactions*, *15*(3), 11–15. doi:10.1145/1353782.1340976

Engelsman, W., & Wieringa, R. (2012). Goal-Oriented Requirements Engineering and Enterprise Architecture. *Requirements Engineering: Foundation for …*, (iv), 306–320. doi:10.1007/978-3-642-28714-5_27

Eriksson, H., & Penker, M. (2000). Business Modeling With UML. *Business Patterns at Work*, 12. doi:978-0471295518

Erl, T. (2005). *SOA: Principles of Service Design. 2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.* doi:10.1109/SAINTW.2003.1210138

Ferreira, M. G., & Wazlawick, R. S. (2011). Software process improvement: A organizational change that need to be managed and motivated. *Proceedings of World Academy of Science, Engineering and Technology*, *74*(2), 301–309. http://www.scopus.com/inward/record.url?eid=2-s2.0-79953280646&partnerID=40&md5=2c1d9d7b1231de33c3e3f5debe8fc48d%5Cnhttp://www.scopus.com/personalization/myprofile.url?zone=TopNavBar&origin=recordpage%5Cnhttp://www.scopus.com/record/display.url?eid=2-s2.0

Garcês, R., Jesus, T. de, Cardoso, J., & Valente, P. (2009). Open Source Workflow Management Systems: A Concise Survey. *2009 BPM & Workflow Handbook*, 179–190.

González, J. L. D. la V., & Díaz, S. (2007). Business process-driven requirements engineering: a goal-based approach. *Proceedings of the 8th Workshop on Business Process Modeling, Development, and Support*, 1–9.

Graham, I. (1996). Task scripts, use cases and scenarios in object oriented analysis. *Object Oriented Systems*, *3*(3), 123–142.

Grudin, I. (1994). Computer-supported cooperative work: History and focus. *Computer*, *27*(5), 19–26. doi:10.1109/2.291294

Grundy, J. (2013). Foreword by John Grundy: Architecture vs Agile: competition or cooperation? In *Agile Software Architecture*. ISBN: 9780124077720

Hagen, V. (2010). An Overview of BPMN 2 . 0 and its Potential Use. *LN Business Information Processing*, *67*. doi:https://doi.org/10.1007/978-3-642-16298-5_3

Hintzen, J., Van Kervel, S. J. H., Van Meeuwen, T., Vermolen, J., & Zijlstra, B. (2014). A professional case management system in production, modeled and implemented using DEMO. *Proceedings of 16th IEEE Conference on Business Informatics*, *1182*. ISBN: 1613-0073

ISO. (2003). *Internacional ISO/IEC 15504-2* (Vol. 2003).

Jacobson, I. (1992). *Object-oriented Software Engineering*. New York, USA: ACM. ISBN: 0-201-54435-0

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). The Unified Software Development Process. *IEEE Software*, *16*, 96–102. doi:10.1109/TSE.2006.59

Joshi, A., Sarda, N. L., & Tripathi, S. (2010). Measuring effectiveness of HCI integration in software development processes. *Journal of Systems and Software*, *83*(11), 2045–2058. doi:10.1016/j.jss.2010.03.078

Karner, G. (1993). Resource estimation for objectory projects. *Objective Systems SF AB*, 1–9. http://si.lopesgazzani.com.br/docentes/marcio/gcm/p_Karner_ResourceEstimationForObjectoryProjects.pdf

Kervel, S. van, Dietz, J. L. G., Hintzen, J., van Meeuwen, T., & Zijlstra, B. (2013). Enterprise Ontology Driven Software Engineering. *Journal of Chemical Information and Modeling*, *53*, 1689–1699. doi:10.1017/CBO9781107415324.004

Kiva. (2017). Kiva - Loans that Change Lives. https://www.kiva.org/

Kluza, K., & Nalepa, G. (2013). Towards Rule-oriented Business Process Model Generation. *Computer Science and Information Systems*, 939–946. ISBN: 9781467344715

Korherr, B., & List, B. (2006). Aligning business processes and software connecting the UML 2 profile for event driven process chains with use cases and components. In *The 18th Conference on Advanced Information Systems Engineering (CAiSE '06)* (Vol. 231).

Luxembourg.

Kreitzberg, C. (1996). Managing for Usability. *Multimedia: A management perspective*.

Kruchten, P. (2004). *The Rational Unified Process: an introduction* (3rd ed.). Addison Wesley. doi:10.1109/ICSE.2002.146346

Laporte, C. Y., Alexandre, S., Irish, T., & Engineering, S. (2008). A Software Engineering Lifecycle Standard for Very Small Enterprises, 129–141.

Larman, G. (2001). Applying UML and Patterns. In *8th Conference on the Advances of Computer Entertainment Technology* (p. 627). Prentice Hall Professional. ISBN: 0130925691

Ling, T.-W., Tompa, F. W., & Kameda, T. (1981). An Improved Third Normal Form for Relational Databases. *ACM Trans. Database Syst.*, *6*(2), 329–346. doi:10.1145/319566.319583

Loniewski, G., Insfran, E., & Abrah, S. (2010). A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development. *Model Driven Engineering Languages and Systems*, (Mdd), 213–227. doi:10.1007/978-3-642-16129-2_16

Martins, P. V., & da Silva, A. R. (2016). ProPAM/static: A static view of a methodology for process and project alignment. *Advances in Intelligent Systems and Computing*, *405*, 47–57. doi:10.1007/978-3-319-26285-7_5

Mazón, J., Pardillo, J., & Trujillo, J. (2007). A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses. *Science*, (ii), 255–264. doi:10.1007/978-3-540-76292-8_31

Memmel, T., Box, D., Box, D., Reiterer, H., & Box, D. (2007). Agile Human-Centered Software Engineering. In *Proceedings of the 21st British HCI Group Annual Conference on\ People and Computers BCS-HCI '07* (pp. 167–175). Lancaster, United Kingdom.

Mittal, K. (2006). Build Your SOA Part 3 The Service-Oriented Unified Process.pdf. https://www.ibm.com/developerworks/library/ws-soa-method3/index.html

Morandini, M., Dalpiaz, F., Nguyen, C. D., & Siena, A. (2014). The Tropos Software Engineering Methodology. *Handbook on Agent-Oriented Design Processes*, 463–490. doi:10.1007/978-3-642-39975-6_14

Morelli, N., & Tollestrup, C. (2006). New Representation Techniques for Designing in a Systemic Perspective. *8th International Conference on Engineering and Product Design Education, E and DPE 2006*, (March 2016), 81–86.

Morgenshtern, O., Raz, T., & Dvir, D. (2007). Factors affecting duration and effort estimation errors in software development projects. *Information and Software Technology*, *49*(8), 827–837. doi:10.1016/j.infsof.2006.09.006

Nair, S., Luis, J., Vara, D., & Sen, S. (2013). A Review of Traceability Research at the Requirements Engineering Conference RE@21. In *Requirements Engineering Conference (RE)* (pp. 222–229). Rio de Janeiro, Brazil: IEEE. doi:10.1109/RE.2013.6636722

Navarre, D., Palanque, P., & Winckler, M. (2009). Task Models and System Models as A Bridge Between Hci and Software Engineering. *Human-Centered Software Engineering, HCI*, 357–385. doi:10.1007/978-1-84800-907-3_17

Nunes, N. (2001). *Object Modeling for User-Centered Development and User Interface Design : The Wisdom Approach*. University of Madeira.

Nunes, N., Constantine, L., & Kazman, R. (2011). iUCP: Estimating Interactive-Software Project Size with Enhanced Use-Case Points. *IEEE Software*, *28*(4), 64–73. doi:10.1109/MS.2010.111

Nunes, N. J., & Cunha, J. F. (1999). A Bridge Too Far: The WISDOM Approach. *Proceedings* *of the Workshop on Object-Oriented Technology*, 283–291. ISBN: 3-540-66954-X

Object Management Group. (2007). The Business Motivation Model, (September), 1–82. http://www.businessrulesgroup.org/second_paper/BRG-BMM.pdf

Ouyang, C., Dumas, M., Van Der Aalst, W. M. P., Hofstede, A. H. M. Ter, & Mendling, J. (2009). From Business Process Models to Process-Oriented Software Systems. *ACM Transactions on Software Engineering & Methodology*, *19*(1), 2:1-2:37. doi:10.1145/1555392.1555395

Papazoglou, M. P., Heuvel, W. Van Den, Papazoglou, M. P., & Heuvel, W. Van Den. (2006). Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, *2*(4), 412–442. doi:10.1504/IJWET.2006.010423

Páscoa, C., & Tribolet, J. (2015). Organizational Operating Systems, an Approach. *Procedia Computer Science*, *64*, 180–187. doi:10.1016/j.procs.2015.08.479

Paternò, F. (1999). *Model-Based Design and Evaluation of Interactive Applications*. London, UK: Springer-Verlag. doi:10.1007/978-1-4471-0445-2

Paternoster, N., Giardino, C., Unterkalmsteiner, M., & Gorschek, T. (2014). Software development in startup companies : A systematic mapping study, *56*, 1200–1218. doi:10.1016/j.infsof.2014.04.014

Patrício, L., Fisk, R. P., & Falcão e Cunha, J. (2008). Designing Multi-Interface Service Experiences. *Journal of Service Research*, *10*(4), 318–334. doi:10.1177/1094670508314264

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, *17*(4), 40–52. doi:10.1145/141874.141884

Phillips, C., & Kemp, E. (2002). In support of user interface design in the rational unified process. *Australian Computer Science Communications*, *24*, 21–27. doi:10.1145/563997.563989

Pine, J., & Gilmore, J. H. (1998). Wellcome to the Experience Economy. *Harvard Business Review*, *76*(4), 97–105. doi:Article

Pütz, C., & Sinz, E. J. (2010). Model-driven Derivation of BPMN Workflow Schemata from SOM Business Process Models. *Enterprise Modeling and Information Systems Architecture*, *5*(2), 1–16. doi:http://dx.doi.org/10.18417/emisa.5.2.4

Ritchey, T. (2015). Principles of Cross-Consistency Assessment in General Morphological Modelling. *Acta Morphologica Generalis*, *4*(2), 1–20. http://www.amg.swemorph.com/pdf/amg-4-2-2015.pdf

Scaled Agile. (2017). Scaled Agile Framework (SAFE). http://www.scaledagileframework.com/

Schwaber, K. (2004). *Agile Project Management with Scrum*. Pearson Education. ISBN: 9780735637900

Shostak, L. (1982). How to Design a Service. *European Journal of Marketing*, *16*(1), 49–63. doi:https://doi.org/10.1108/EUM0000000004799

Sousa, K., Doyens, P., Mendonça, H., Rogier, E., & Vandermeulen, J. (2008). User Interface Derivation from Business Processes : A Model-Driven Approach for Organizational Engineering. *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, 553–560. doi:10.1145/1363686.1363821

Stamper, R. (2013). On developing organisational semiotics as an empirical science: the need for scientific method and rigorous debate. *Proceedings of the Fourteenth International Conference on Informatics and Semiotics in Organisations*, 1–13. ISBN: 9789898565518

Systems, S. (2017). Enterprise Architect. www.sparxsystems.com

The Open Group. (2011). *TOGAF ® Version 9.1 Enterprise Edition - An Introduction.*

The Open Group. (2012). *An Introduction to ArchiMate, an Open Group Standard.* www.opengroup.org/archimate

The Standish Group. (2013). Chaos Manifesto 2013: Think Big, Act Small. *The Standish Group International*, 1–52. http://www.standishgroup.com

The Standish Group. (2014). The Standish group: the chaos report. *Project Smart*, 16. doi:10.1016/S0895-7061(01)01532-1

Tryggeseth, E., & Nytro, I. (1997). Dynamic traceability links supported by a system architecture description. In *Proc. of ICSM 1997* (pp. 180–187). doi:10.1109/ICSM.1997.624244

Ullah, A., & Lai, R. (2011). Modeling Business Goal for Business / It Alignment Using Requirements Engineering. *Journal of Computer Information Systems*, *51*(3), 21–28. doi:10.1080/08874417.2011.11645482

Valente, P. (2009). *Goals Software Construction Process*. Saarbrucken, Germany: VDM Verlag Dr. Muller. ISBN: 978-3639212426

Valente, P., Aveiro, D., & Nunes, N. (2015). Improving Software Design Decisions towards Enhanced Return of Investment. *Proceedings of the 17th International Conference on Enterprise Information Systems*, 388–394. doi:10.5220/0005383803880394

Valente, P., & Sampaio, P. N. M. (2007). Process Use Cases: Use cases Identification. *ICEIS 2007 - 9th International Conference on Enterprise Information Systems, Proceedings, ISAS*, 301–307.

Valente, P., Silva, T., Winckler, M., & Nunes, N. (2016a). Bridging Enterprise and Software Engineering Through an User-Centered Design Perspective. In *Web Information Systems Engineering (WISE)* (Vol. 2, pp. 463–477). doi:10.1007/978-3-319-26190-4

Valente, P., Silva, T., Winckler, M., & Nunes, N. (2016b). The Goals Approach: Enterprise Model-Driven Agile Human-Centered Software Engineering. *Human-Centered and Error-Resilient Systems Development*, *9856*, 261–280. doi:10.1007/978-3-319-44902-9_17

Valente, P., Silva, T., Winckler, M., & Nunes, N. (2017). The goals approach: Agile enterprise driven software development. *Lecture Notes in Information Systems and Organisation*, *22*, 201–219. doi:10.1007/978-3-319-52593-8_13

Van Lamsweerde, A. (2000). Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, *26*(10), 978–1005. doi:10.1109/32.879820

VersionOne.com. (2017). 11th Annual State of Agile Report. *VersionOne Agile Annual Report*, 1–16. doi:10.1093/jicru/ndl025

Villiers, D. de. (2003). Using the Zachman Framework to Assess the Rational Unified Process Overview of the Zachman Framework, 1–10.

Winckler, M., Cava, R., Barboni, E., Palanque, P., & Freitas, C. (2015). Usability Aspects of the Inside-in Approach for Ancillary Search Tasks on the Web. In *Human-Computer Interaction - INTERACT 2015* (pp. 211–230). Bamberg, Germany: Springer International Publishing. doi:10.1007/978-3-319-22668-2_18

Yu, E. (1995). Modelling Strategic Actor Relationships for Business Process Reengineering. *International Journal of Cooperative Information Systems*, *4*(02n03), 125–144. doi:10.1142/S0218843095000056

Zachman, J. (2008). The Concise Definition of The Zachman Framework. Zachman International.

Zachman, J. A. (1986). A Framework for Information Systems Architecture. *IBM Systmes Journal*, *26*(3), 454–470. doi:10.1147/sj.263.0276

Zidisha. (2017). Zidisha - Person to Person Microlending. https://www.zidisha.org/

Zikra, I. (2014). Integration of Enterprise Modeling and Model Driven Development: A Meta-Model and a Tool Prototype. *Universitetsservice AB*, 1–145. http://www.diva-portal.org/smash/record.jsf?pid=diva2:719487

Zukowski, J. (2013). *The Definitive Guide to Java Swing*. *Apress* (3rd ed., Vol. 53). Apress. doi:10.1017/CBO9781107415324.004

Appendix A – The Goals Conceptual Approach

This appendix presents theoretical ideas which are useful to help understanding the logic behind the language presented in this thesis.

The Goals Approach is based on the idea that the more a given "being" is aware if its own capabilities, the more the probabilities it will have of achieving *goals* in the most efficient and effective way. The capabilities refer to the structure and readiness of the structure for action, and the act of achieving *goals* refers to the action promoted over the structure in order to fulfil *goals* that improve its survival capabilities. The logic and structure derived from the idea are presented as follows, using the Goals constructive concepts using italic font. Applied to the enterprise domain, the enterprise is the "being" which is being considered, where efficiency refers to the *resources* used to achieve a *goal*, and effectiveness refers to the number of times that a *goal* is achieved against how many times it is tried, and performance is the efficiency in achieving successful *goals*. The *resources*, which are internal to the system, are represented by *business concepts*, which are instantiated in order to produce the *data* of the Information System, and are subject to *regulations* which are expressed by composed (or complex) *business concepts*.

Humans interact with the enterprise system using Interaction Spaces where they can carry on their User Tasks, as long as the *regulations* (Business Rules) which they are subject to, are guaranteed by the Interaction Space functioning. The Interaction Spaces of the system are physical, software-based, or combined, meaning that when humans interact with each other in real physical places, the Interaction Space that they occupy must provide at least the *data* (Data Entities) needed in order carry on the Users Tasks, independently of the used support e.g. paper or Database. This is seen as the minimum business-support implementation possible, so that users can exchange, produce and keep the *data* generated from their labor. If the *data* exchange is subject to control under the enterprise defined *regulations*, this means that a second level of implementation, where they are applied in an automated way or by means of human control, exists. In the first case an automated software Information System, in the second, a user which responsibility of action is under the guidance of the enterprise (a collaborator) must ensure that the *regulations* are applied, recurring to the existing physical *resources*. An Information System support for the user's activity can be total or partial i.e. the information exchange can be totally carried out inside the Information System, or the human users must recur to other means in order to be able to fulfil their *tasks*. If the automation is complete (of 100%), then the interaction between the users is necessarily remote (even if they are in the same physical space). In the case of a remote interaction, each user benefits from one Interaction Space perspective (called as an Aggregation Space), which provides all the tools and *data* which are necessary in order to carry on his *task*(s), including *data* exposure and *regulations* application. This happens in the same way that when a user interacts with another in person (in 0% automation), each one has his own perspective based on the information that he can retain (senses) and process (brain and body) from his environment, the outside external real world in that moment. In this case, every business *regulation* is processed in each user's mind, and for that reason, not subject to automated enterprise verification, needing human control in order to ensure that they are satisfied, a control which is left to the enterprise needs, as users are supposed to carry on their *tasks* with *wisdom* and according to the enterprise's *ideology*.

With these premises, users cooperate with each other in order to attain the enterprise's *goal* in which they are working on, being aware (the enterprise collaborator), or not necessarily (e.g. the customer or third party *actor*) of it (the enterprise *goal*). The human activity that is carried out in order to attain a *goal* is the Business Process organization that is promoted by the enterprise. The Business Process and *goal* concepts cover both, the daily operational routine of the enterprise, and the business process (or *goal*) monitoring activity, even if executed by a single *actor* and related to strategical (e.g. market-oriented) *goals*. From the cost of the human and system activity it is possible to calculate the *task* efficiency which is matched against the produced *resources*, meaning that when the enterprise "being" uses less *resources* in order to achieve a *goal*, then the better is the (enterprise) performance.

Appendix B – Interaction Patterns

The Goals Approach is presented based on a specific interaction design pattern by default which enables attaining a level of detail that allows the maximization of the software development performance. I.e. the more the system is detailed, the more modeling investment in terms of man-hour is needed, but as a return, the implementation will be probably be faster, as *Software Developers* will have less doubts concerning each component implementation. This is a trade-off that should be analyzed in terms of its long-term impact, since the Information System has a tendency to grow with the automation of BPs, and the complexity that may be generated from systems with a low level of organization may become (the complexity) unmanageable, even if only a reduced part of the enterprise BPs are automated. For example, a small enterprise may have 15 BPs, and by implementing 5 with e.g. 25 UTs, the generated system may easily have more than 100 components. These are figures from which a patterns organization of the system may arise, or contrarily, it may already be sufficient in order to generate architectural chaos.

Thus, each interaction design pattern has advantages and disadvantages in the business and software perspectives. The interaction design pattern used to illustrate the Goals targets a high level of modularization, independently of the used implementation framework. However, depending on time and budget, the *Software Architect* may decide, that the option of a smaller implementation, and a lower level organization may apply, reason why we present the possible interaction design patterns which may be elaborated with Goals. The default pattern includes the option for a User Interface organization that first receives the *data*, presents it, and upon user command saves the *data* to the Database. This aims software development performance, by means of a separation of concerns which is achieved by means of the distinction between User Interface-oriented and Database-oriented Business Logic components. This pattern is called as the "Read and Save", and since the user interaction is specified in two level (User Intentions and Interaction, Steps 4 and 5 of the Goals method), it is further defined as "Read and Save – Interaction Design" presented in Figure 88.

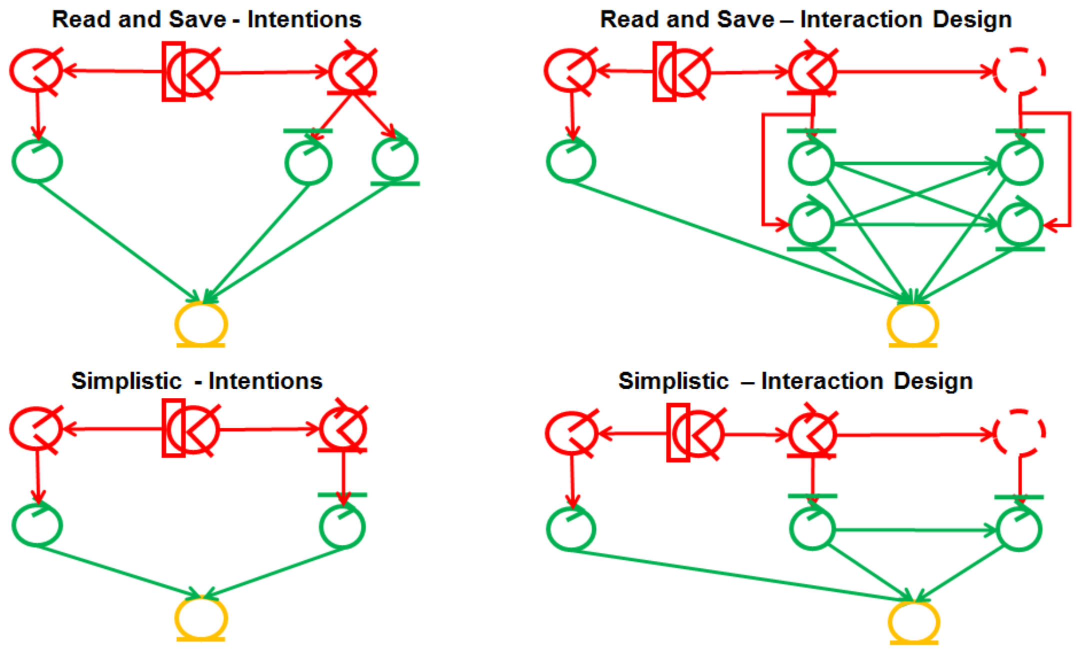

Read and Save - Intentions
Read and Save – Interaction Design
Simplistic - Intentions
Simplistic – Interaction Design

Figure 88. Design Patterns for Human-Computer Interaction.

The standard pattern defines User Interface (UISR) and Database System Responsibilities (DBSR) that will serve as an electronic interface of access to the more specific UISRs and DBSRs that will be elicited in the Interaction Modeling. The advantage of this patterns is that it defines a hierarchy of functions that ensures modularization and facilitates the reuse of the system. The disadvantage is that it involves an investment in terms of design and implementation that will only be returned in the long term, with the benefits of the organization. An alternative to this patterns which has the advantage of reducing the investment time, is not designing the user interaction and leaving the responsibility of implementation to the UISRs and DBSRs which are identified in the Task Model. This is an approach which is close to the well-known method "just do it" (absence of method). It may be applied to small improvements, but it has the disadvantage of reducing the reuse capabilities of the implemented SRs without restructuring (refactoring [Fowler, 2000]) parts which are already implemented. Since the system is designed based only upon the User Intentions, it is named as "Read and Save – Only Intentions". If complementarily there is the need to save more time not making the design distinction between UISR and DBSR, then the implementation is also possible by implementing the SRs whether they are generated from the Task of the Interaction models (Steps 4 or 5, the last of the Analysis and the First of the design, respectively). These systems, namely if produced without the detail of the user interaction, may become unmanageable when the system grows if a strong nomenclature that facilitate component reuse is not applied. They are named as "Simplistic", and further defined as "Intentions" and "Interaction Design". Notice that an implementation of a Simplistic patterns that predicts an homogeneity of the SRs in which the communication happens in real time between the User Interface and the Database, it would be "cooperative work" [Grudin, 1994], since if two users are using the same User Interface of the system for the same BP execution using the same AS at the same time, then they would be doing cooperative work, and if they would do it using two different AS, then when one user would change *data*, the other one would see it if that *data* would make part of the other user's AS. If the Software Architecture would be considered without the User Interface, then it could be considered as Software Oriented Architecture (SOA), in which the Web Services Interfaces and Web Services Atomic Functions could be derived from the elaboration of the Task and Interaction models using e.g. personas instead of *actors*, as presented in Figure 89.

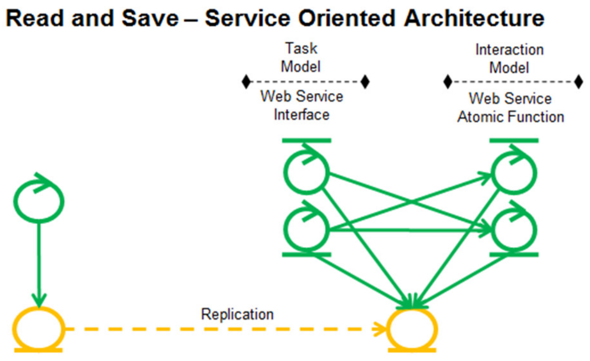

Read and Save – Service Oriented Architecture
Task Model
Web Service Interface
Interaction Model
Web Service Atomic Function
Replication

Figure 89. Design Pattern: Service Oriented Architecture (SOA).

References

[Fowler, 2000] Fowler, M., et al., Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2000.

[Grudin, 1994] Grudin, J.: Computer-supported cooperative work: history and focus. In: Computer, V. 27, pp. 19-26. (1994)

Appendix C – "iKiosk" Project Diagrams

We now present the remaining diagrams for the "iKiosk" project.

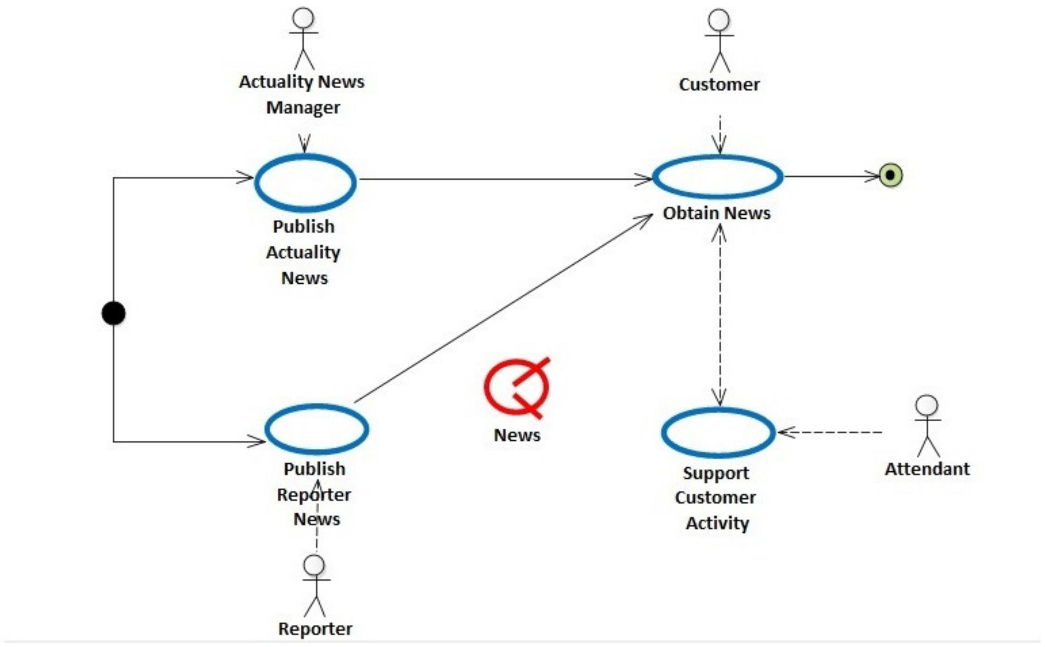


Figure 90. Business Process "Decoy".

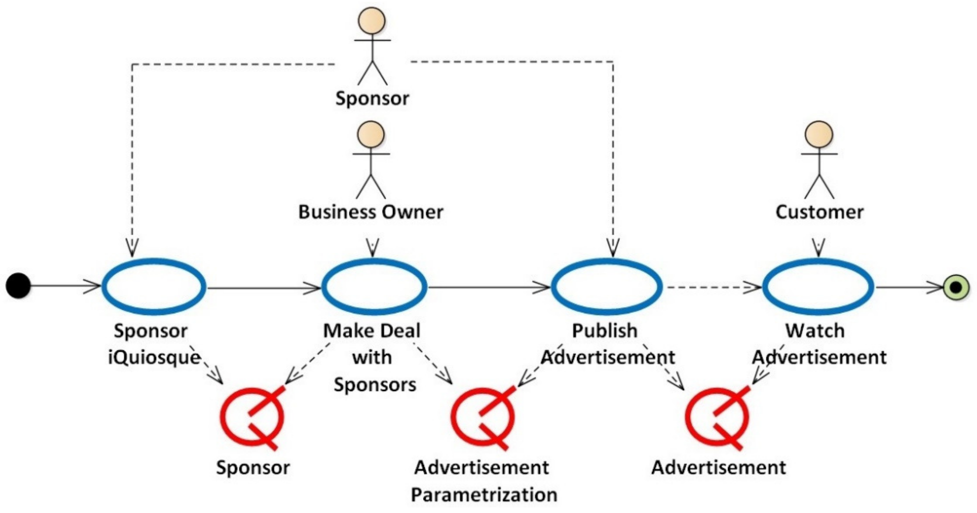


Figure 91. Business Process "Publish Advertisement".

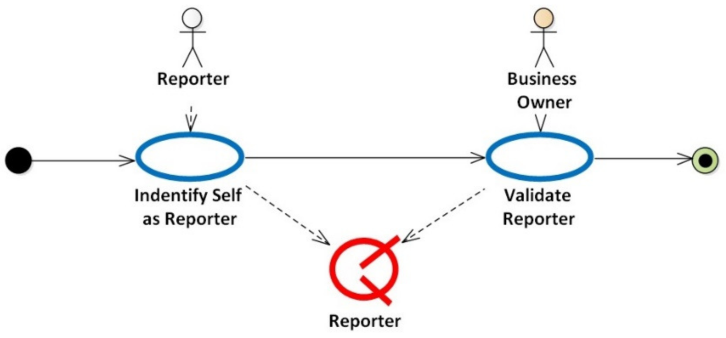


Figure 92. Business Process "Validate Reporter".

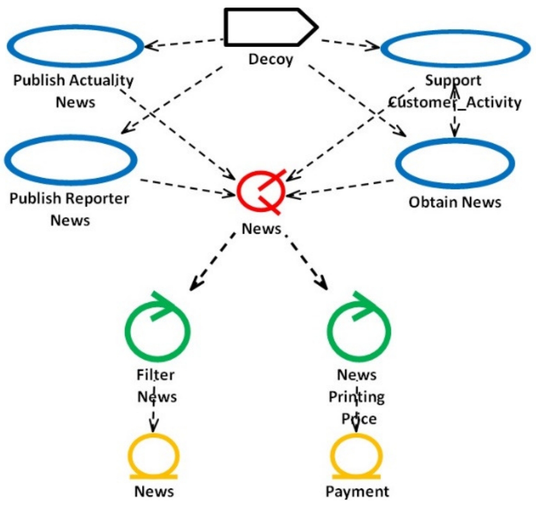


Figure 93. Enterprise Structure "Decoy".

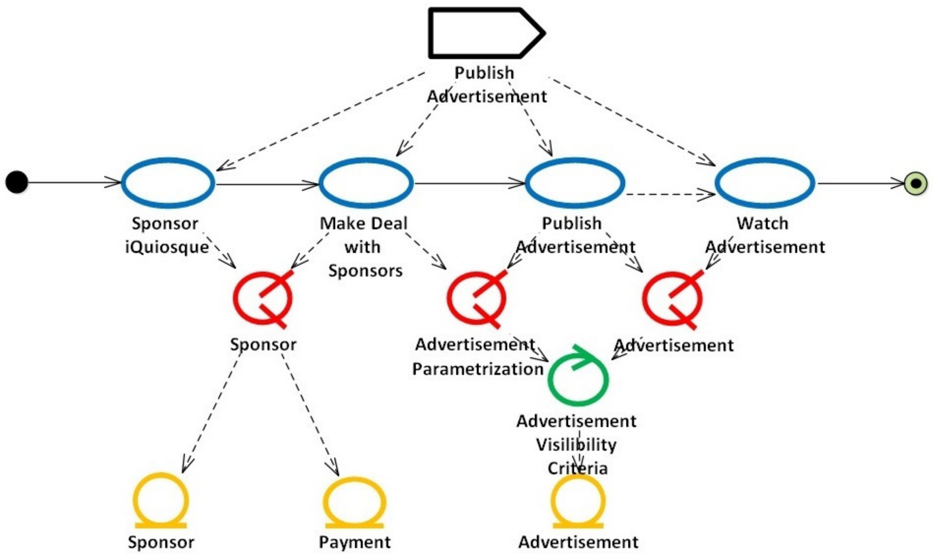


Figure 94. Enterprise Structure "Publish Advertisement".

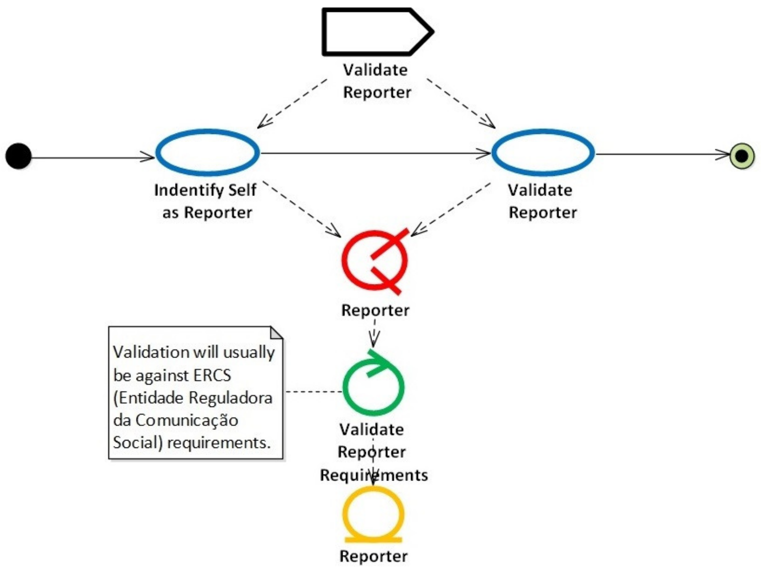

Validate Reporter
Indentify Self as Reporter
Validate Reporter
Reporter
Validation will usually be against ERCS (Entidade Reguladora da Comunicação Social) requirements.
Validate Reporter Requirements
Reporter

Figure 95. Enterprise Structure "Validate Reporter"

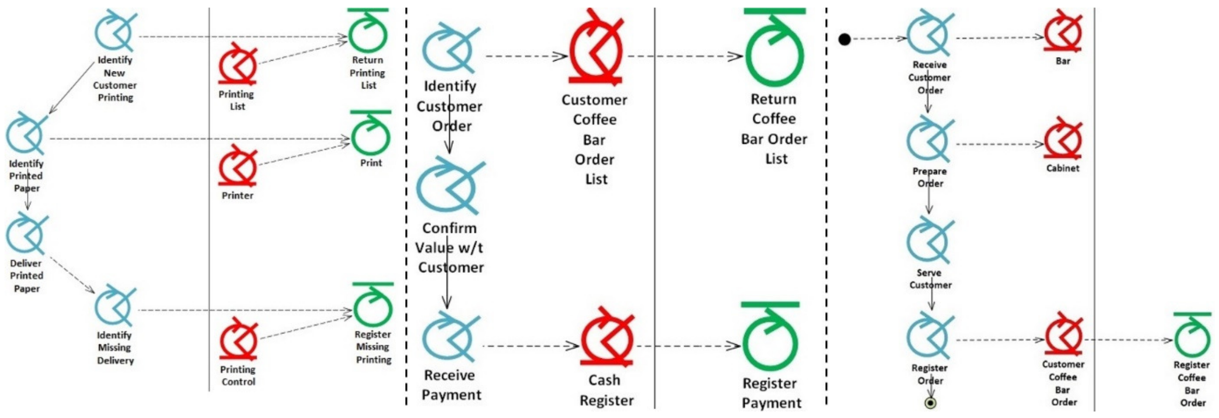


Figure 96. Task Model “Deliver Printing”, “Receive Payment”, “Serve Coffee” UT.

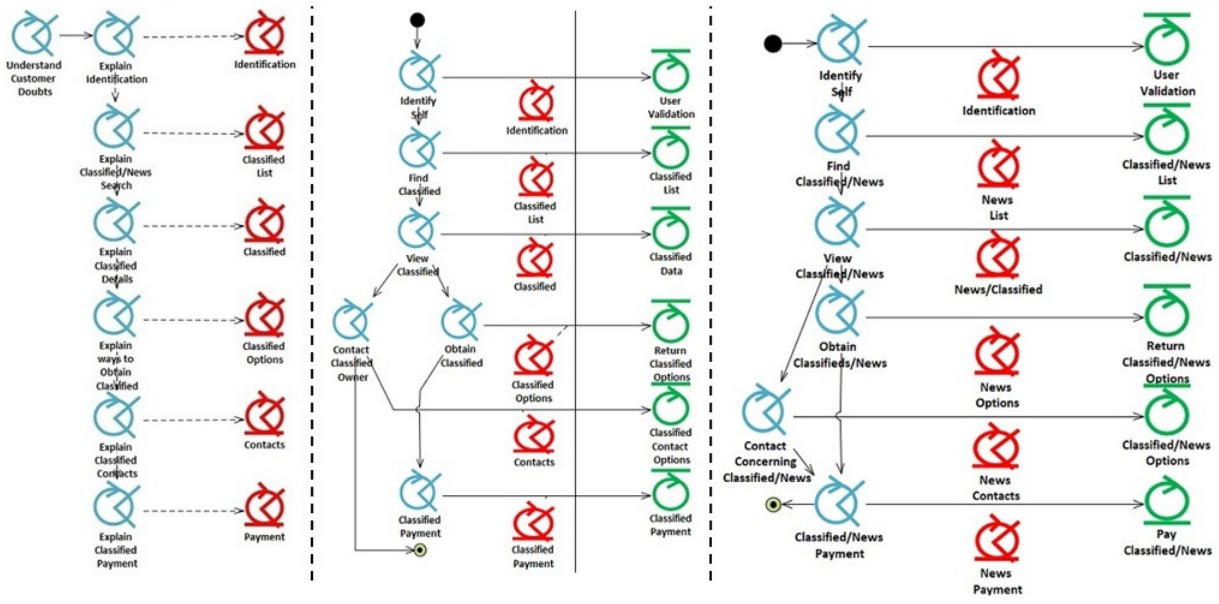


Figure 97. Task M. “Support Customer Activity”,“Obtain Classified”,“Obtain News” UT.

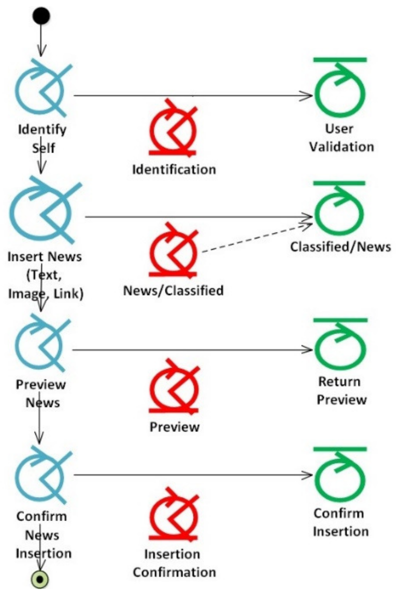


Figure 98. Task Model “Publish Reporter News”

Appendix D – "Star Project" Diagrams

We now present the remaining diagrams for the "Star Project".

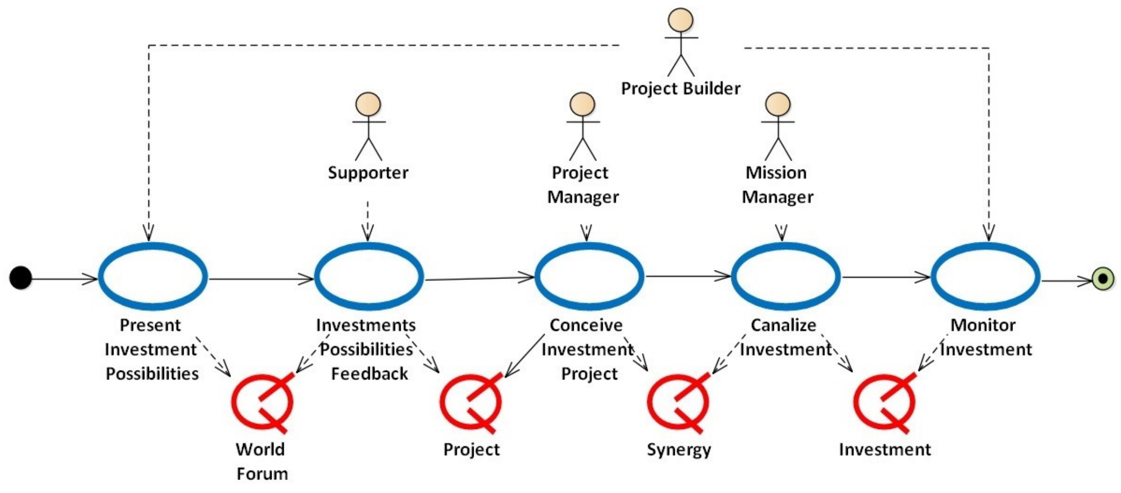


Figure 99. Business Process "Conceive Investment Project".

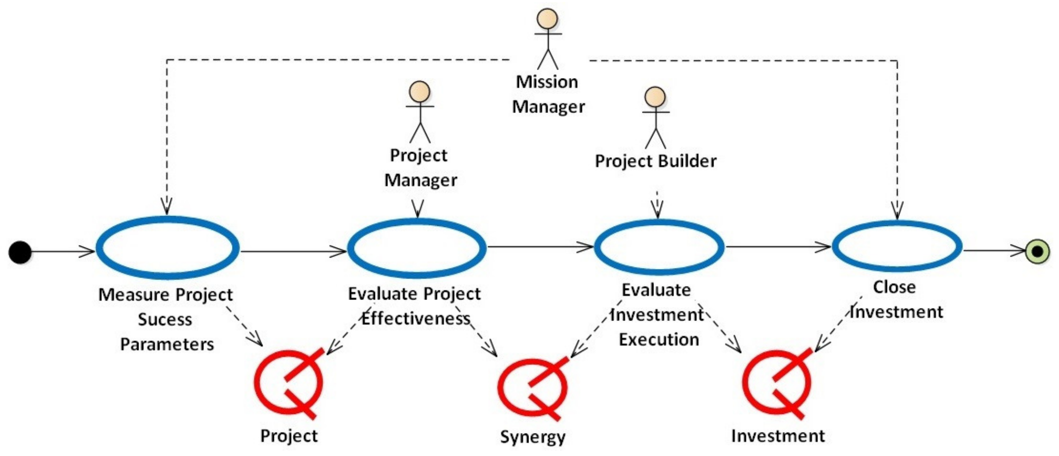

Mission Manager
Project Manager
Project Builder
Measure Project Sucess Parameters
Evaluate Project Effectiveness
Evaluate Investment Execution
Close Investment
Project
Synergy
Investment

Figure 100. Business Process "Evaluate Investment".

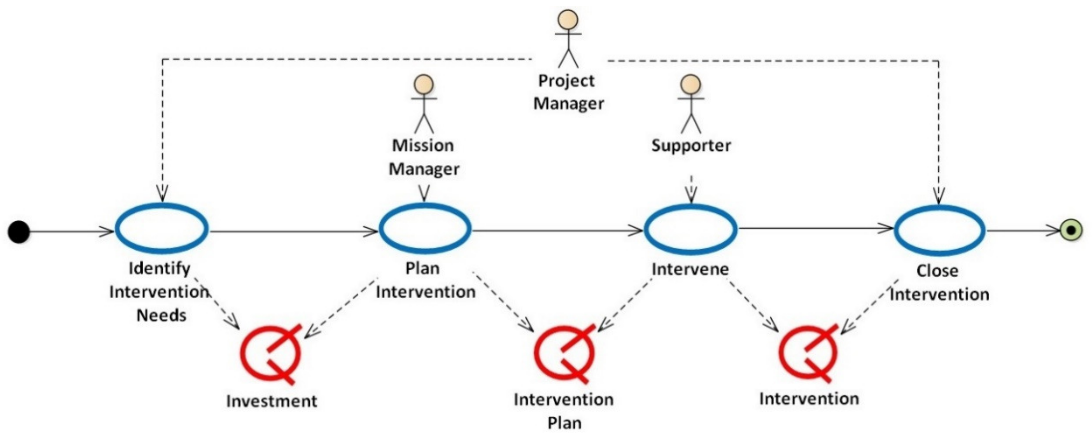


Figure 101. Business Process "Support Investment".

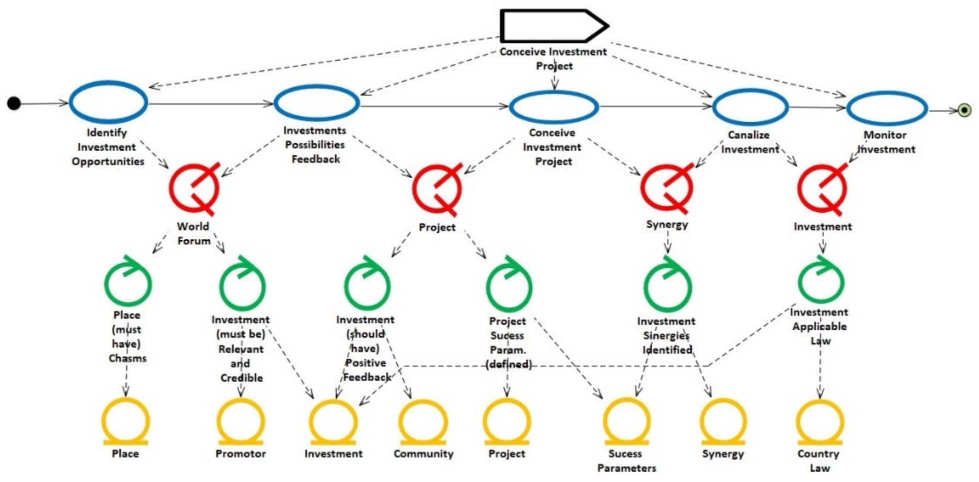

Conceive Investment Project
Identify Investment Opportunities
Investments Possibilities Feedback
Conceive Investment Project
Canalize Investment
Monitor Investment
World Forum
Project
Synergy
Investment
Place (must have) Chasms
Investment (must be) Relevant and Credible
Investment (should have) Positive Feedback
Project Sucess Param. (defined)
Investment Sinergies Identified
Investment Applicable Law
Place
Promotor
Investment
Community
Project
Sucess Parameters
Synergy
Country Law

Figure 102. Enterprise Structure “Conceive Investment Project”.

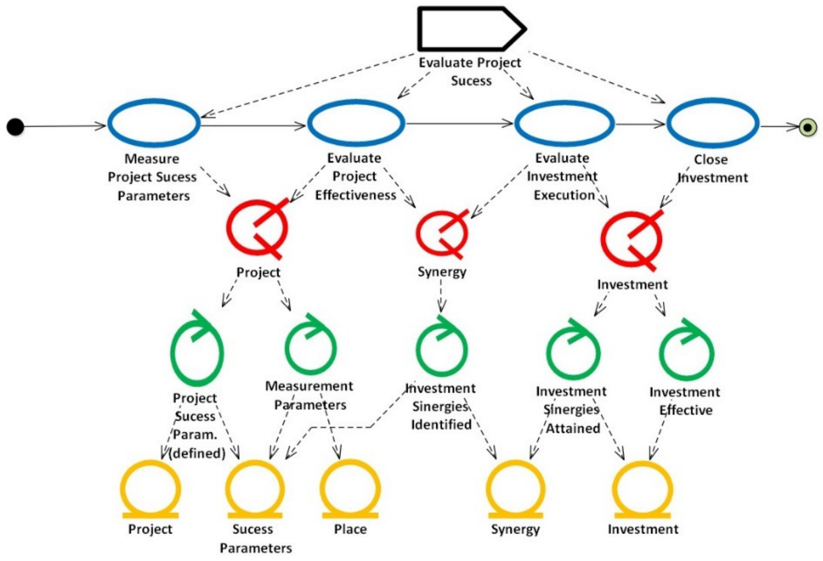

Evaluate Project Sucess
Measure Project Sucess Parameters
Evaluate Project Effectiveness
Evaluate Investment Execution
Close Investment
Project
Synergy
Investment
Project Sucess Param. (defined)
Measurement Parameters
Investment Sinergies Identified
Investment Sinergies Attained
Investment Effective
Project
Sucess Parameters
Place
Synergy
Investment

Figure 103. Enterprise Structure “Evaluate Investment”.

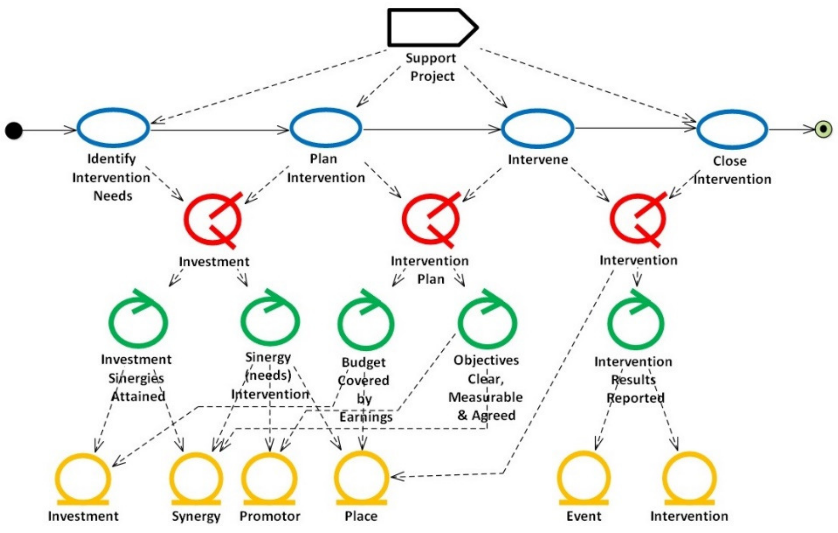
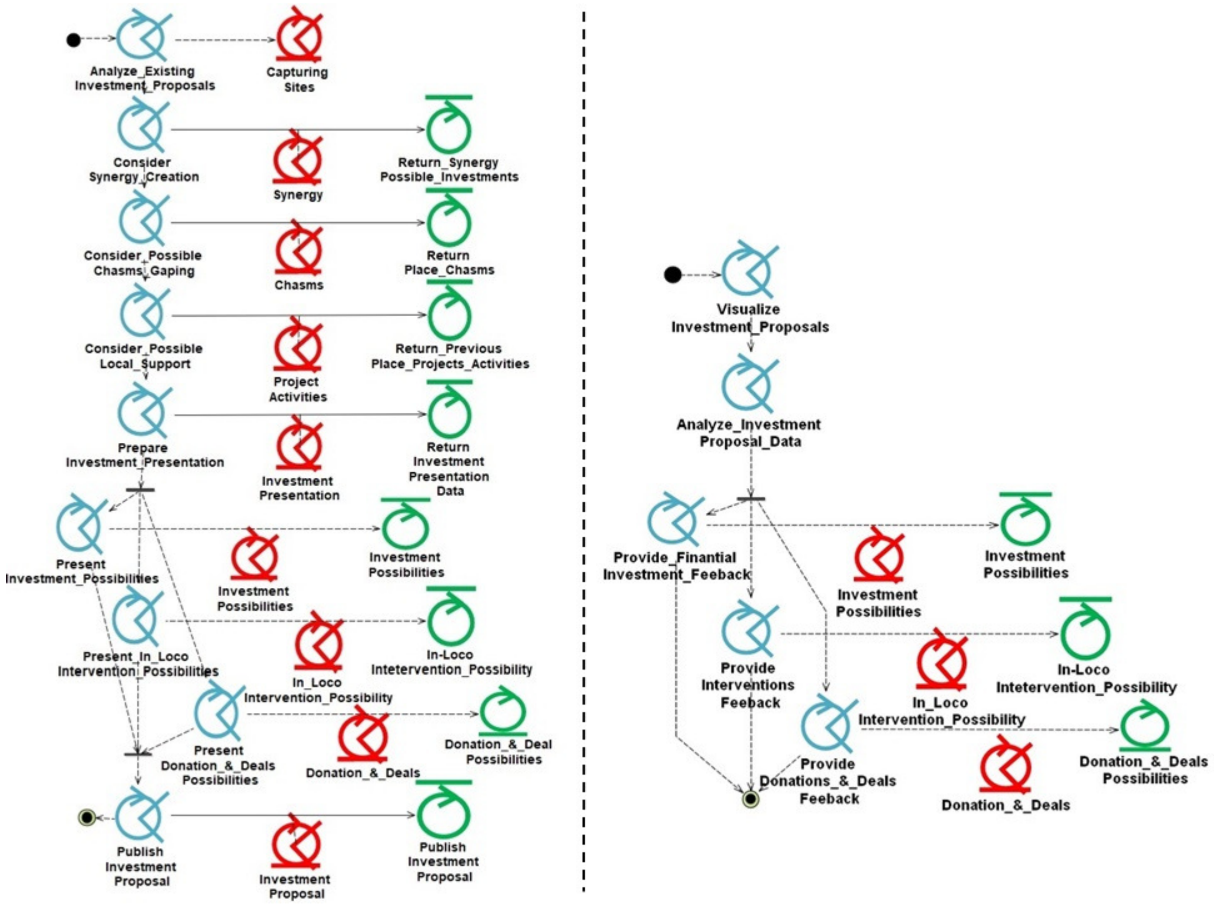
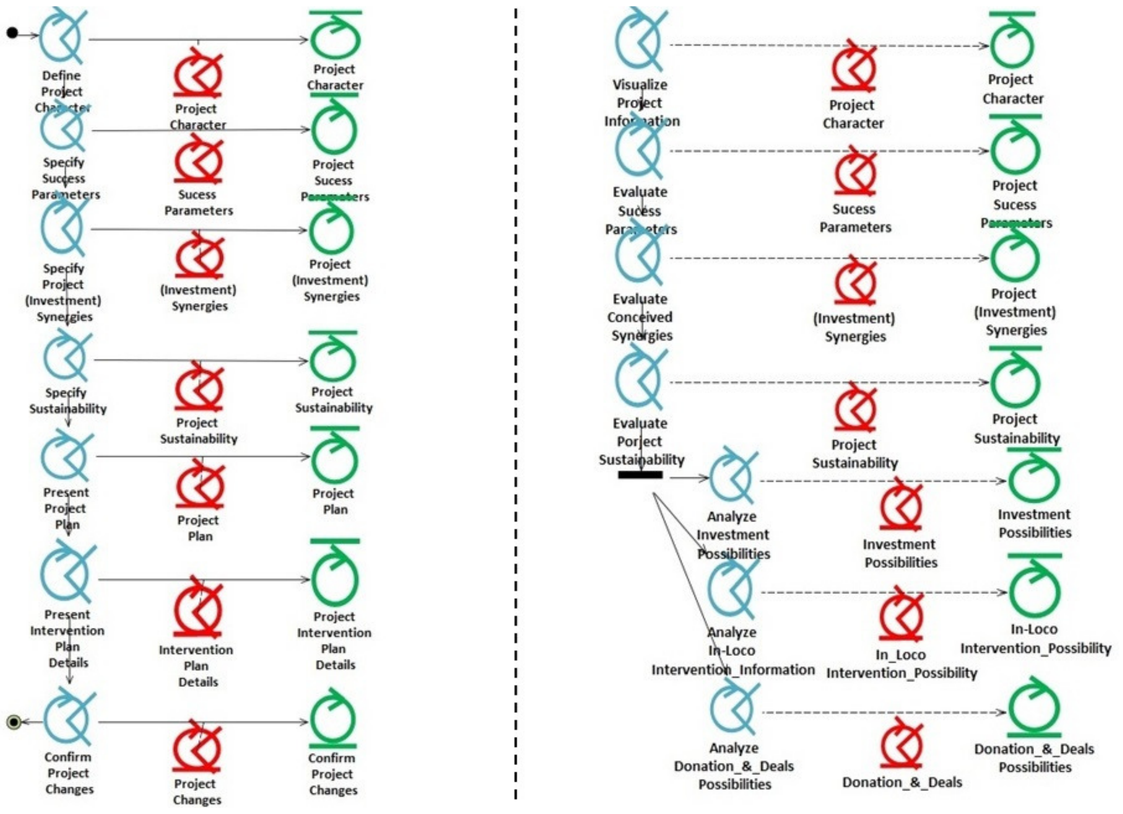

Support Project
Identify Intervention Needs
Plan Intervention
Intervene
Close Intervention
Investment
Intervention Plan
Intervention
Investment Sinergies Attained
Sinergy (needs) Intervention
Budget Covered by Earnings
Objectives Clear, Measurable & Agreed
Intervention Results Reported
Investment
Synergy
Promotor
Place
Event
Intervention

Figure 104. Enterprise Structure “Support Project”.



Analyze_Existing Investment_Proposals
Capturing Sites
Consider Synergy_Creation
Synergy
Return_Synergy Possible_Investments
Consider_Possible Chasms_Gaping
Chasms
Return Place_Chasms
Consider_Possible Local_Support
Project Activities
Return_Previous Place_Projects_Activities
Prepare Investment_Presentation
Investment Presentation
Return Investment Presentation Data
Present Investment_Possibilities
Investment Possibilities
Investment Possibilities
Present_In_Loco Intervention_Possibilities
In_Loco Intervention_Possibility
In-Loco Intetervention_Possibility
Present Donation_&_Deals Possibilities
Donation_&_Deals
Donation_&_Deal Possibilities
Publish Investment Proposal
Investment Proposal
Publish Investment Proposal
Visualize Investment_Proposals
Analyze_Investment Proposal_Data
Provide_Finantial Investment_Feeback
Investment Possibilities
Investment Possibilities
Provide Interventions Feeback
In_Loco Intervention_Possibility
In-Loco Intetervention_Possibility
Provide Donations_&_Deals Feeback
Donation_&_Deals
Donation_&_Deals Possibilities

Figure 105. TM "Present Investment Possibilities"|"Investment Possibilities Feedback".



Define Project Character
Project Character
Project Character
Specify Success Parameters
Sucess Parameters
Project Sucess Parameters
Specify Project (Investment) Synergies
(Investment) Synergies
Project (Investment) Synergies
Specify Sustainability
Project Sustainability
Project Sustainability
Present Project Plan
Project Plan
Project Plan
Present Intervention Plan Details
Intervention Plan Details
Project Intervention Plan Details
Confirm Project Changes
Project Changes
Confirm Project Changes
Visualize Project Information
Project Character
Project Character
Evaluate Sucess Parameters
Sucess Parameters
Project Sucess Parameters
Evaluate Conceived Synergies
(Investment) Synergies
Project (Investment) Synergies
Evaluate Porject Sustainability
Project Sustainability
Project Sustainability
Analyze Investment Possibilities
Investment Possibilities
Investment Possibilities
Analyze In-Loco Intervention_Information
In_Loco Intervention_Possibility
In-Loco Intervention_Possibility
Analyze Donation_&_Deals Possibilities
Donation_&_Deals
Donation_&_Deals Possibilities

Figure 106. TM "Present Project" | "Support Project".

Glossary

A Nossa
Universidade

Colégio dos Jesuítas
Rua dos Ferreiros - 9000-082, Funchal

Tel: +351 291 209400
Fax: +351 291 209410
Email: gabinetedareitoria@uma.pt