Departament of Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

State of the Art

for the Systematic Construction and Analysis of *i** Models for assessing COTS-Based Systems Development

Author:

Gemma Grau Colom

Barcelona, November 2006

Abstract

This document presents the state of the art related with the systematic construction and analysis of *i** models for assessing COTS-based systems development. The fist section presents an overview of the Component-Based Systems (CBS) development processes. As components are part of the architecture of the system, the second section introduces the evaluation of software architectures. The *i** framework has been proved useful on the representation and evaluation of software architectures, including those containing COTS, the third section presents the *i** framework and some other requirements engineering techniques. As the *i** framework is agent-oriented, and so, the fourth section presents an overview of agent-oriented paradigm. Finally, as CBS development is an activity that seldom takes place from the scratch, we can tackle it as a process reengineering activity, because of that, section 5 outline the main issues in business process reengineering.

Table of Contents

1. Overview of the CBS Development Process

1.1. Introduction to the CBS development paradigm

The growing importance of the COTS-Based Systems (hereafter, CBS) development paradigm is mainly due to the benefits it provides in terms of quality, development time and cost; especially when delivering large, complex systems. However, CBS development also implies some risks such as the ones related to complications in the development or post-deployment situations [Abts-et-al. 2000]. Thus, development of CBS requires the application of well-disciplined systematic methodologies, especially in the activities of selection and evaluation.

Nowadays, software development processes for CBS are largely studied and there is a significant body of knowledge that identifies issues and proposes frameworks for improving them. For instance, in [Brownsword-et-al. 2000] the changes required to address the CBS development are identified taking into account real-life lessons and a framework is articulated for organizing the new and changed process elements. In [Morisio-et-al. 2000] a report is done about adopted COTS-based processes and a new one is proposed.

According to COCOTS (Constructive COTS integration cost model) [Abts-et-al. 2000] the initial integration cost in the development of CBS is mainly due to the effort needed to perform:

1. Candidate COTS component assessment, where COTS candidates are determined based on the functional requirements (capability offered), performance requirements (timing and sizing constraints) and non-functional requirements (cost, training, installation, maintenance, reliability).
2. COTS component tailoring.
3. The development and testing of any integration or clue code needed to plug a COTS component into a larger system, and
4. Increased system level programming and testing due to volatility in incorporated COTS components.

From this list it can be observed that, in CBS development, early evaluation and selection of candidate COTS components is a key aspect on the system development lifecycle. As remarked in [Ncube-Maiden 1999], the selection of suitable COTS products is often a non-trivial task and requires careful consideration of multiple criteria. Additionally, COTS are not designed to operate isolated and, when selecting a COTS, the dependencies and interactions with the other components of the system, has also to be considered [Franch-Maiden 2003].

Because of that, integration cost has also to be taken into account when informing a COTS selection process. More precisely, a good estimate of integration cost can inform the decision of using or not an specific COTS solution, the selection of the best COTS

products, and to determine the amount and type of glueware that needs to be built [Yakimovich-et-al. 1999].

1.2. The COTS Selection Process

According to [Kunda-Brooks 1999], there are three phases of COTS software selection:

- **Evaluation criteria definition.** This process essentially decomposes the requirements for the COTS into a hierarchical criteria set. The criteria include component functionality (what services are provided), other aspects of a component's interface (such as the use of standards) and quality aspects that are more difficult to isolate, such as components reliability, predictability, and usability.
- **Identification of candidate components**. Also known as alternatives identification, the identification of candidate components involves the search and screening for candidate COTS that should be included for assessment in the evaluation phase.
- **Evaluation of the criteria for these candidates**. There are currently three strategies to COTS evaluation:

  Progressive filtering is a strategy whereby a COTS product is selected from a larger set of potential candidates, in which products that do not satisfy the evaluation criteria are progressively eliminated from the products list.

  In keystone strategy, products are evaluated against a key characteristic such as a vendor or type of technology.

  In the puzzle assembly model, a valid COTS solution will require fitting the various components of the system together.

The evaluation criteria definition phases assume that the requirements for the COTS have already been obtained. However, when doing CBS development, it is impossible to find a single product that meets all the requirements obtained from the stakeholders, and the requirements acquisition process has to take into account the capabilities offered by the different vendors. To tackle this problem, quality models [Franch-Carvallo 2003] are useful as: the attributes for a certain kind of component are described in a hierarchical way; COTS products are evaluated according to that description; requirements are stated using the attributes as a basis; and, then, the more appropriate product is selected by matching.

Nevertheless, if the organization needs more capabilities than the ones offered by a single COTS supplier, the CBS may have to integrate several COTS which have to interoperate together to provide this functionality. The impact of using different COTS components is expected to vary with the domain: for business applications a large, pervasive COTS product may be used to deliver one or more requirements (e.g., MS Office, Oracle, Netscape, etc.); when a de facto standard product is used, the COTS capabilities determine the requirements [Abts-et-al. 2000]; finally, for embedded real time of safety critical domains, the COTS components are expected to be smaller and require large amounts of glue code to integrate the set of components [Chung-Subramanian 2001].

As work has progressed in the area of COTS research, specific solutions have been proposed to address some of these COTS selection issues. Table 3.1 presents a summary

with some of the most important proposals and the phases they mainly address, more information about this methods can be found in:

- OTSO: Off-The-Shelf Option Method [Kontio 1996]
- STACE: Social-Technical Approach to COTS Evaluation framework [Kunda-Brooks 1999]
- PORE: Procurement-Oriented Requirements Engineering technique [Maiden-Ncube 1998].
- COSTUME: *CO*mposite *S*of*T*ware system q*U*ality *M*odel d*E*velopment [Carvallo-Franch-Grau-Quer 2004]
- MATE: Middleware Architecture and Technology Evaluation [Gorton-Liu 2002]
- The CARE (COTS-Aware Requirements Engineering) process [Chung-Kooper 2004]

Table 3.1. Summary of methodologies dealing with COTS selection.

Methods	Main Focus	Requirements Acquisition	Evaluation Criteria Definition	Candidate Components Identification	Evaluation of Candidates
OTSO	Hierarchical evaluation criteria definition	-	✓	~	✓
STACE	Social and organizational issues	-	✓	-	-
PORE	Iterative process based on templates	✓	✓	~	✓
MATE	Selection of COTS middleware products	✓	✓	✓	✓
COSTUME	Non-functional Requirements	✓	✓	✓	✓
CARE	agent and goal-oriented methodology	✓	~	✓	-

(✓) addresses the issue fully (~) deals with the issue but not fully (-) does not deal with the issue

During the COTS selection processes, functional and non-functional requirements have to be taken into account. Additionally, as COTS components are continually evolving in response to the market, methodologies that cost-effectively manage the use of those evolving components have to be adopted [Abts-et-al. 2000]. As pointed out in [Alves-Castro 2001], COTS selection processes deals with the following 4 dimensions:

- **Domain Coverage**. The components have to provide all or part of the required capabilities, which are necessary to meet core essential customer's requirements.
- **Time restriction.** Software companies usually operate in a very rigid development schedule, on which their competitiveness depends. Selection is a time consuming activity, where a considerable amount of time is necessary to search and screen all the potential COTS candidates.
- **Cost rating.** The available budget is a very important variable. The expenses when selecting COTS products will be influenced by factors such as: license acquisition, cost of support, adaptation expenses, and maintenance prices. [Boehm-et-al. 1998] provides an economic model for estimating the cost of COTS-based system development.

  Vendor guaranties. An important aspect to be considered in the selection activity is to verify the technical support provided by the vendor. Some issues have to be taken into account, for example: vendor reputation and maturity, number and kind of

applications that already use the COTS, clauses characteristics of the maintenance licenses.

1.3. Interoperability Issues

In CBS development, COTS have to be integrated in order to provide its capabilities. According to [Bao-Horowitz 1996] integrations approaches are:

- White Box: users must have access to the source code of third-party software in order to modify it and add in whatever functionalities needed for the integration.
- Grey Box: requires that the third-party vendors foresee the integration requirements and provide needed APIs or message interfaces when they build software for the users.
- Black Box: relies on command-line options and it gets input and output only at the starting and ending time of the application.

The usage of COTS software restricts somehow the use of this integration approaches because, most of the times, COTS product source code is not available to the application developer, and the future evolution of the COTS product is not under the control of the application developer [Abts-et-al. 2000]. This fact constraints the integration of COTS products onto a Black Box approach. However if in-house developed software or open-code components are interacting with the COTS, other approaches may be applied.

These lead to two different options to achieve interoperability, which are mentioned in [Sauer-et-al. 2000]:

- Heterogeneous software is transformed into an agreed format in order to interoperate in a common domain; or
- Interfaces of heterogeneous software are adopted into a common interoperable ground, leaving the implementations in their original language, domain, and bindings.

The first option applied to COTS means that only the products providing a certain interoperability format are adopted (sometimes vendors may adapt its interfaces to required software in a grey box approach). The second option is a pure black approach where glue code and middleware products are used. However, these solutions cannot be so easily tackled, as the major reasons that cause difficulty in COTS integration still being, according to [Bao-Horowitz 1996]:

- Unique constraints of third-party software (inadequacy of integration interfaces, closeness for the system architecture, lack of access to the source code…)
- Interactive nature of most of software applications today (graphical user interface, incremental input and output, interpretative execution…)
- Increasing emphasized new integration requirements (broad framework applicability, general encapsulation scheme, strong support for system evaluation, component reusability, end-user programmability…)

1.4. COTS Integration Support

The difficulties in COTS integration make the integration phase of CBS development at a high risk. Because of that, COTS integration support is needed. Some general considerations about COTS interoperability can be found in [Yakimovich-et-al. 1999], including a method for estimating its cost. More general foundations about interoperability problems can be found in [Guo 2000]. Finally, case studies are also useful [Balk-Kedia 2000; Warboys-et-al. 2005].

In order to solve interoperability issues, [Yakimovich-et-al. 1999] mentions the interaction between components as the origin of the integration problem, which are general for all kind of components (including COTS). Four types of interaction are defined:

- Component-platform interactions. A component must be executed somewhere. It can be either a real processor with an operating system for binary executables, or a virtual one. If an executable program was compiled for one type of CPU, it will need an emulator or a code converter in order to run it on another CPU.
- Component-hardware interactions. A component can interact directly with hardware writing-reading from ports. If the port's numbers are different from what is expected by the component, the component must undergo some modification.
- Component-user interactions. A component's user interface requirements may also change. For example, a component can have its messages in one language, when the system requires another language.
- Component-software interactions. A component almost always interacts with other software components, and there can be mismatches between the components. A set of possible mismatches between components: representation, communication, packaging, synchronization, semantics, control, etc.

In the component-software interaction level, "every component is designed with assumptions concerning its interactions, and the assumptions strongly depend on the particular architecture" [Yakimovich-et-al. 1999]. Architectural assumptions that can cause mismatches in component interaction are defined in [Garlan-et-al. 1995]. These assumptions are about the nature of the components (infrastructure, control model and data model), the nature of the connectors (protocols and data model), the global architecture structure and the construction process.

In order to deal with these assumptions, [Yakimovich-et-al. 1999] proposes a set of variables to represent inter-component interactions, which are: component packaging, type of control, information flow, synchronization and component binding. Based on them, the main architectural styles can be classified. Table 3.2 shows the results of this classification.

Table 3.2. Common architectural styles and their classification according to a relevant set of variables. Obtained from [Yakimovich-et-al. 1999].

	Packaging	**Control**	**Information Flow**	**Synchronization**	**Binding**
Pipes and Filters	Not relevant	Not relevant	Data	Not relevant	Dynamic
Main program and subroutine	Not relevant	Centralized	Control	Synchronous	Static

OO Systems	Not relevant	Centralized	Control	Synchronous	Dynamic
Communicating processes	Not relevant	Decentralized	Not relevant	Not relevant	Not relevant
Event Systems	Not relevant	Decentralized	Control	Not relevant	Dynamic
Blackboards	Not relevant	Not relevant	Data	Not relevant	Static
Chiron-2 (C2)	Depends on language supported	All types	Data (messages)	All types	Dynamic
CORBA	Depends on languages supported	Decentralized	Control (RPC)	All types	Run-time dynamic
COM	Depends on languages supported	Decentralized	Control	All types	Run-time dynamic

A similar evaluation of approaches to software interoperability can be found in [Guo 2000]. Despite this evaluated approaches are not explicitly focus on COTS (it evaluates Wrappers, Data Mediators, Data Replicators, Data Translators, Messaging, ORBs and JINI), the evaluation criteria can be applied to a COTS context (in terms of Performance, Reliability, Speed to field, Extendibility, Maintainability, Security).

In order to deal with COTS integration, also published case studies are useful. For instance [Balk-Kedia 2000] presents a COTS integration case study where the CBS was composed by COTS already available in the organization. As a result, the selection of COTS and the testing of the products are simplified, whilst the integration phase has become the most costly. A proposal for flexible COTS integration is in [Warboys-et-al. 2005], and consist of a framework for adapting software in dynamic environments such as the ones constructed with COTS products. The framework proposes an architecture that allows interoperability between the different components by means of a specific Architecture Description Language.

1.5. Remarks

The mentioned process and methodologies deal with to different phases of the CBS development. Some of them tackle the evaluation of COTS components for choosing the most suitable, whilst the others provide guidelines for the COTS integration process. Despite some selection methods take into account COTS interoperability, as far as we know, none of them takes into account the interoperability issues in a direct way.

However, we strongly believe that taking into account interoperability issues in the COTS selection assessment phase, will certainly provide a better assessment of the architecture of COTS and, as a result, the cost of the integration phase may be reduced. For doing so, we need to take the COTS architecture into account and provide a way to represent it and evaluate it. To address this issue, the next section provides some considerations about architecture representation and evaluation.

2. Evaluating Software Architectures

2.1. The Importance of Software Architectures

"Software architecture is a growing field of research and practice within software engineering. This is mainly due to the fact that there is proof of evidence of causal connections between design decisions made in the architecture and the qualities and properties that result downstream in the system or systems that follow from it" [Clements-et-al. 2002].

Although being widely studied, there is no standard definition about what is software architecture. However, two definitions are commonly cited:

> *The software architecture of a program of computing system is the structure of structures of the system, which comprises software components, the externally visible properties of those components, and the relationship among them.*
>
> L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice. Addison-Wesley, 1998. Page 279.

> *Abstractly, software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on this patterns. In general, a particular system is defined in terms of a collection of components and interactions among those components.*
>
> M. Shaw and D. Garlan. Software Architecture – perspectives on an emerging discipline. Prentice-Hall, Inc., 1996. Page 1.

Software architecture deals with the description of the structure of a software system's and, as pointed out in [Bass 1998], embodies the most fundamental and hardest to change design decisions. Due to this fact and according to [Clements-et-al. 2002], software architecture has a strong determining effect on: the system's realization of quality attributes (e.g., performance, modifiability, availability or security); the work breakdown structure of the developments which is determined by the modules or subsystems within the architecture; and the planning for a software product line which is based upon a common architecture and a set of shared assets.

Another point to be taken into account is that early design decisions made in the architecture result in constraints on an implementation. For example, [Clements-et-al. 2002] states that such constraint is the choice of commercial components that can be easily integrated. Although interoperability protocols exist, choosing any commercial component will affect the choice of the other components that will be employed.

Finally, in large, complex, software intensive systems, software architecture is especially important as it provides a link between the different elements of the project.

For instance, [Clements-et-al. 2002] mentions that software architecture is a vehicle for communication among stakeholders; the manifestation of the earliest design decisions; and a reusable, transferable abstraction of a system.

Software architectures capture early design decisions that reflect major quality concerns, including functionality. On the other hand, during system design quality requirements has to be taken into account because early decisions have an impact on the final qualities and properties of the system. According to [Clements-et-al. 2002], architectural decisions can be analysed in the context of the goals and requirements that are levied on systems that will be build from it.

In order to obtain the benefits of architecture analysis specification languages and analysis techniques are needed [Medvidovic-Taylor 1997]. The following sections introduce several issues for describing architectures and evaluate software architectures are analysed.

2.2. Architecture Description Languages

According to [Clements-et-al. 2002], the main factors of software architecture's success are: improved communication from one stakeholder to another; assisted analysis by describing the right information; and being build from the blueprints that the architectural description represents. Consequently, *specification languages are needed to demonstrate properties of a system upstream, thus minimizing the costs of errors and to provide abstractions adequate for modelling a large system, while ensuring sufficient detail for establishing properties of interest* [Medvidovic-Taylor 1997].

In order to answer to that need, a large number of architecture description languages (ADLs) has been proposed. However, since it's not the goal of this state of the art to report and compare all the existing languages as this has already been done in [Clements 1996; Medvidovic-Taylor 1997; Medvidovic-Taylor 2000]. Instead, we provide an enumeration of the elements of an ADL and some guidelines for distinguish ADLs from other languages.

As stated in [Vestal 1993], "an ADL for software applications focuses on the high-level structure of the overall application rather than the implementation details of any specific source module". Thus, according to [Medvidovic-Taylor 1997; Medvidovic-Taylor 2000], ADLs typically subsumes a formal semantic theory and provide both a concrete syntax and a conceptual framework for modelling software system's conceptual architecture. Concretely, [Coyette 2003] proposes a state-of-the-art ontology that contains the essential aspects of a system architecture that any ADL should be able to specify. The proposed definitions are based on the most relevant architectural characteristics and requirements in literature. The following concepts are defined:

- **Components.** Units of computation or data stores. A component in architecture may be as small as a single procedure or as large as an entire application. It may require its own data and/or execution space, or it may share it with other components.
- **Interfaces**. Set of interaction points among itself and the external world. All ADLs support specification of component interfaces, although they differ in the terminology and the kinds of information they specify.

- **Connectors.** They are used to model interactions among components and the rules that govern those interactions. The notion of connection varies among languages.
- **Configurations**. Connected graphs of components and connectors that describe architectural structure. Configurations are needed to determine whether appropriate components are connected, their interfaces match, and their combined semantics result in desired behaviour. Descriptions of configurations enable assessment of concurrent and distributed aspects of architecture, adherence to design heuristics and style constraints.
- **Constraints.** Properties or assertions about a system, the violation of which will render the system unacceptable to one or more stakeholders. Constraints ensure adherence to intended component uses, enforce usage boundaries, and establish intra-component dependencies.
- **Hierarchical compositions**. They represent architectures as single components in other larger architectures.

In order to distinguish ADLs for other high-level design languages, [Medvidovic-Taylor 1997; Medvidovic-Taylor 2000] adds the requirement that ADLs have to model the configurations explicitly. Hence, those languages where configurations are modelled implicitly through interconnection information are not considered ADLs, and fall into the category of implicit configuration languages. On the other hand, in-line and explicit configuration languages are ADLs, and model configurations explicitly, with the difference that in-line configuration languages specify connector information only as part of the configuration whilst explicit configuration languages model both components and connectors separately from configurations.

Using the requirement that configurations had to be modeled explicitly, high-level design notations, programming languages, object-oriented modelling notations and formal specification languages are not considered ADLs. However, they may serve as useful tools when dealing with certain aspects of the architecture. For instance, this is the case of Use Case Maps (UCM) and *i**. In [deBruin-vanVliet 2002] UCM had been used for generating alternative software architectures because they provide stubs where the behaviour of a system can be varied statistically at construction time as well as dynamically at run time. The *i** framework has also been used as a modelling languages to assess architecture evaluation in [Chung-Nixon-Yu 1999; Bastos-Castro 2003] as it allows to focus more on the business process and non functional requirements of the application.

As a conclusion, we mention the fact that each ADL focuses on different aspects of the architectural description and analysis. Because of that, none of them has widely adopted [Clements-et-al. 2002; Medvidovic-Taylor 2000].

2.3. Software Architecture Evaluation

According to [Clements-et-al. 2002], early design decisions of architecture allows or preclude nearly all of the system's quality attributes. Thus, architecture evaluation is becoming an accepted engineering practice because of the enormous risks that architecture represents in a development project.

The evaluation of software architectures may be done at any cycle of the architecture's lifetime. However, the more earlier it is performed, the more helpful is for dealing with opportunities and risks. Because of that, the classical application of architecture evaluation takes places after the specification of the architecture, but before beginning its implementation. An *early* evaluation is done when the evaluation is made before the architecture is fully specified; whilst a *late* evaluation takes place when the implementation is complete (usually when an organization inherits some sort of legacy system).

Depending on the stage of the architecture where the evaluation takes place, different techniques can be applied. Questioning techniques are appropriate in every state of the architecture lifecycle. On the other hand, measuring techniques require some artefact to be measured. Finally, hybrid techniques combine elements of both approaches and their application depends on the applicability conditions of the specific technique. Table 3.3 shows a classification and comparison table showing the main characteristics of each approach, which have been obtained from [Clements-et-al. 2002]. Other works that compare software architecture analysis methods are [Babar-Gorton 2004; Dobrica-Niemelä 2002].

Table 3.3. Evaluation techniques compared. Obtained from [Clements-et-al. 2002].

Technique	Quality Attribute(s) Covered	Approach(es) Used	When applied
Questioning Techniques	Allow to investigate any area of project in virtually any state of readiness.		
Questionnaires and Checklists	Various	Predefined domain-specific questions	Can be used to prompt architect to take certain design approaches, or any time thereafter.
Scenario-based methods	Various: either non-run-time attributes such as modifiability or run-time attributes such as security.	System specific scenarios to articulate specific quality attribute requirements; scenario walkthroughs to establish system response	When architecture design is complete enogh to allow scenario walk-thoughts
SAAM	Modifiability, Functionality		
ARID	Suitability of Design		
SNA	Security		
Measuring Techniques	Requires the existence of some artifact to measure.		
Metrics	Various: often emphasize modifiability and reliability	Static analysis of structure	After architecture has been designed
Simulations, Prototypes, Experiments	Various: often emphasize performance, functionality, usability	Measurement of the execution of an artifact	After architecture has been designed
RMA	Performance oriented to real-time systems	Quantitative static analysis	After the process model has been built and process-to-processor allocations have been done.
ADLs	Various: tend to concentrate on behaviour and performance	Simulation, symbolic execution	When architectural specifications are complete
Hybrid Techniques	Combine elements from questioning and measuring techniques.		
SPE	Performance	Scenarios and quantitative statistical analysis	When performance constraints have been assigned to architectural components.
ATAM	Not oriented to any particular quality attributes, but historically emphasizes modifiability, security, reliability and performance	Utility trees and brain-stormed scenarios to articulate quality attribute requirements; analysis of architectural approaches to identify sensitivities, trade-off points, and risks.	After the architectural design approaches has been chosen.

Definitions of some of the attributes that can be evaluated in software architectures are stated in table 3.4.

Table 3.4. Quality attributes for evaluation purposes. Definition from [Clements-et-al. 2002].

Availability
Availability is the proportion of time the system is up and running. It is measured by the length of time between failures as well as how quickly the system is able to resume operation in the event of failure.
Conceptual Integrity
Conceptual integrity is the underlying theme or vision that unifies the design of the system at all levels. The architecture should do similar things I similar ways. Conceptual integrity is exemplified in an architecture that exhibits consistency, has a small number of data and control mechanisms, and uses a small number of patterns throughout to get the job done.
Functionality
Functionality is the ability of the system to do the work for which it was intended. Performing a task requires that many or most of the system's components work in a coordinated manner to compete the job. It can be understood in terms of how the architectural pieces interact and cooperate with each other to perform the system's work.
Modifiability
Modifiability is the ability to make changes to a system quickly and cost effectively. It is measured by using specific changes as benchmarks and recording how expensive those changes are to make.
Performance
Performance refers to the responsiveness of the system – the time required to respond to stimuli (events) or the number of events processed in some interval of time. Performance qualities are often expressed by the number of transactions per unit time or by the amount of time it takes to complete a transaction with the system. Performance measures are often cited using *benchmarks*, which are specific transaction sets or workload conditions under which the performance is measured.
Portability
Portability is the ability of the system to run under different computing environments. These environments can be hardware, software, or a combination of the two. A system is portable to the extent that all of the assumptions about any *particular* computing environment are confined to one component (or at worst, a small number of easily changed components). If porting to a new system requires change, then portability is simply a special kind of modifiability.
Reliability
Reliability is the ability of the system to keep operating over time. Reliability is usually measured by mean time to failure. If reliability is important, then the architecture needs to provide redundant components with warm or hot restart protocols among them.
Security
Security is a measure of the system's ability to resist unauthorized attempts at usage and denial of service while still providing its services to legitimate users. Security is categorized in terms of the types of threats that might be made to the system. If Security is a consideration, then you need to pay attention to inter-component communication and data flow and perhaps introduce special components (such as secure kernels or encrypt/decrypt functions) or impose authentication protocols between processes.
Subsetability
Subsetability is the ability to support the production of a subset of the system. While this may seem like an odd property of an architecture, it is actually one of the most useful and most overlooked. Subsetability can spell the difference between being able to deliver nothing when schedules slip versus being able to deliver a substantial part of the product. Subsetability also enables incremental development, a powerful development paradigm I which a minimal system is made to run early on and functions are added to it over time until the whole system is ready. Subsetability is a special kind of variability mentioned above.
Variability
Variability is how well the architecture can be expanded or modified to produce new architectures that differ in specific, preplanned ways. Variability mechanisms may be run-time (such as negotiating on the fly protocols), compile-time (such as setting compilation parameters to bind certain variables), build-time (such as including or excluding various components or choosing different versions of a component), or code-time mechanisms (such as coding a device driver for a new device). Variability is important when the architecture is going to serve as the foundation for a whole family of related products, as in a product line.

As it is proved that evaluating architecture at the early phases is very effective to assess system development, this work focus in evaluating architectures at an early stage. In this situation, scenario-based architecture evaluation is the most adequate. There are several kinds of scenarios that can be used to clarify different aspects of the architecture:

- **Use case scenarios.** Come up from the description of the interaction between the user and the competed running system. They are used to obtain non-functional requirements.
- **Growth scenarios.** Represent typical anticipated change to a system. "Thus, they concern the present set of requirements as well as possible extensions or changes. The latter are especially useful to assess the architecture with respect to structural aspects such as flexibility or modifiability" [deBruin-vanVliet 2002].
- **Exploratory scenarios.** They expose the limits or boundary conditions of the current design, exposing possibly implicit assumptions.

The goal of this thesis is not to describe all the scenario-based architecture evaluation methods, but due to its relevance two of them are explained:
- SAAM (Software Architecture Analysis Method) [Kazman-et-al. 1994]; and
- ATAM (Architecture Trade-off Analysis Method) [Clements-et-al. 2002].

The Software Architecture Analysis Method (SAAM) [Kazman-et-al. 1994] evaluates software architectures against the desired software attributes and permits the comparison of different software architectures with respect to given properties. SAAM has to be applied once a high-level design of the architecture has been made, so the software architecture design, the business drivers and quality requirements are the main inputs of the method. The six activities of SAAM are: 1) scenario development; 2) software architecture description; 3) scenario classification and prioritization; 4) individual scenario evaluation; 5) scenario interaction; and 6) overall evaluation. In the case of comparing multiple software architectures, weightings are assigned to scenarios in order to determine the overall ranking of the different software architectures. SAAM evaluates scenarios by mapping each scenario onto the software architecture, and checking if the software architecture supports or not the scenario. If the scenario is not supported, the cost of accommodating this scenario is estimated by counting the number of required changes. Scenario interaction analysis reveals if many indirect scenarios affect the same component, a sign of poor separation of concern.

The Architecture Trade-off Analysis Method (ATAM) [Clements-et-al. 2002] has superseded, in some aspects, SAAM. ATAM permits the analysis of software architectures capabilities with respect to multiple quality attributes; and helps to make trade-offs between competing attributes. Although ATAM is applicable during any stage of the software development, it is most effective when applied to the final version of software architecture. The inputs for applying ATAM are: business goals, software specifications and software architecture description. ATAM provides a whole framework for the application of the method. Table 3.5 contains the main steps of the ATAM method and the actions performed, techniques used and main inputs and outputs of each step. Notice that the nine activities of ATAM are repeated in two phases: in the first one only selected stakeholders are involved, whilst in the second one a wide range of stakeholders are requested.

Table 3.5. The process model of ATAM.

Phase	Step	Actions	Resources used	Outputs
Phase 1: Initial Evaluation				
	Step 1	**Present the ATAM**	ATAM presentation viewgraphs	-
	Step 2	**Present the business drivers**	System overview presentation documents, if available from client	Business goals Quality Attributes of Interest
	Step 3	**Present the architecture**	Sample quality attribute characterization	Summary of architecture presentation
	Step 4	**Identify the architectural approaches**	Catalogue of architectural approaches and architectural styles	architectural approaches
	Step 5	**Generate the quality attribute utility tree**	Sample utility tree	Quality attributes and prioritized scenarios
	Step 6	**Analyze the architectural approaches**	Analysis of an architectural approach template	Risks, nonrisks, sensitivity points and tradeoff points
Phase 2: Complete Presentation				
	Step 1-6	**Recapitulation of phase I**	-	Understanding of phase I outcomes
	Step 7	**Brainstorm and prioritize scenarios**	Scenarios and utility tree from step 5.	Prioritized scenarios Augmented quality tree
	Step 8	**Analyse the architectural approaches**	Analysis on how architectural components involved affect each scenario	Risks, sensitivities and tradeoff points
	Step 9	**Present Results**	Template for presentation results	-

ATAM does not prescribe any specific evaluation techniques and uses various theoretical models of the quality attribute communities, applies qualitative reasoning heuristics, architectural patterns, and several kinds of scenarios.

One of the techniques used by ATAM is the application of architectural patterns and styles, which are especially useful for generating alternative architectures. Architectural patterns and styles represent partial design structures by grouping structural and behavioural aspects with the quality properties provided. Architectural patterns are described in terms of its components, connectors, topology, and constraints. This situates them at a low lever of representation of the architecture than the one needed in this document, some references about them may be found in [Klein-Kazman 1999].

In most of the evaluation methods, quality requirements are used as a basis for evaluating the architecture. Consequently, as the requirements engineering community is moving towards a goal-oriented approach [vanLamsweerde-et-al. 1992], there are also some evaluation proposals that take into account, not only the requirements, but also the goals and rationale behind them. An example of goal-oriented architecture evaluation is [Chung-Nixon-Yu 1999]. The method proposes to take into account the dependencies between quality goals and architectural styles. On the one hand, organizational and stakeholders relationship is taken into account by using the *i** framework [Yu 1995] to model and reasoning about strategic relationships. On the other hand, quality requirements are represented and addressed during architectural design by applying the NFR framework [Chung-Nixon-Yu 2000].

Standing on a similar basis, [Kolp-et-al. 2001; Kolp-et-al. 2003; Bastos-Castro 2003] also use the *i** framework to describe the organizational relationships among the different agents of a system, and the NFR framework [Chung-Nixon-Yu 2000] to

evaluate alternative architectures. The domain of application is different, as it is meant to be applied to evaluate the organizational structure of multi-agent architectures.

In REACT [Franch-Maiden 2003] software architectures are also modelled with the *i** framework and a more formal treatment involving metrics is proposed to evaluate system properties over the modelled system in order to inform multiple component selection. The method proposed in [deBruin-vanVliet 2002] uses Use Case Maps (UCM) to model scenario-based architecture descriptions. Candidate architectures are generated, evaluated and refined if needed. This process is assessed by the achievement of non-functional requirements.

2.4. Remarks

Software architecture is a consolidated discipline that has proven to be useful in assessing application development, reducing costs and risks, and increasing communication among the different parts involved. From the study of the different evaluation techniques and architecture description languages, we can observe that most of them concentrate on different facets of architectural description and analysis.

Scenario-based analysis of architectures appears a useful technique for analysing architecture at an early stage. As architectures are evaluated against quality requirements, some goal-oriented approaches are beginning to get used in order to represent and evaluate architectures. This shows that not all the issues have been addressed yet. As stated in [Chung-Nixon-Yu 1999], one key task that remains a difficult challenge for practitioners is how to proceed form requirements to architectural design. The use of a goal-oriented requirements approach by representing architectures in the *i** framework is beginning to get used [Chung-Nixon-Yu 1999, Franch-Maiden 2003, Bastos-Castro 2003].

The next section gives a deeper view on the *i** framework.

3. The *i** Framework

3.1. Goal-Oriented Requirements Engineering

Requirements Engineering (hereafter, RE) research has increasingly recognized the leading role played by goals in the RE process [vanLamsweerde 2001]. In early RE approaches, the focus was on eliciting and documenting the requirements, but not the rationale behind them. As understanding the reasons why a system is being developed is a critical success factor in projects, goal driven approaches focus on this issues. So, in Goal-Oriented RE the relationship between the requirements and their goals is represented explicitly.

A goal is an objective to be achieved by the system under consideration. Goal formulation refers to intended properties to be ensured and, as so, goals may be formulated at different levels of abstraction, ranging from high-level, strategic concerns to low-level, technical concerns.

There are many reasons why goals are so important in the RE process [vanLamsweerde 2001]. Their main benefits are concerned with aspects such as achieving requirements completeness, avoiding irrelevant requirements, explaining requirements to stakeholders, structuring requirements, detecting requirements conflicts, requirements evolution and traceability, identifying requirements, and driving refinement and abstraction.

As a result of their recognized benefits, goals have become the focus of a whole stream of research on goal modelling, goal specification, and goal-based reasoning for multiple purposes, such as requirements elaboration, requirements verification or conflict management, and under multiple forms, from informal qualitative to formal.

Goal-based reasoning reviews how goals are used in basic activities such as requirements elicitation, elaboration, verification, validation, explanation, and negotiation; particularly for difficult aspects such as conflict management, requirements deidealization, and alternative selection [vanLamsweerde-et-al. 1992].

A summary and comparison of several of the most widespread goal-oriented approaches is already available in [Green 1994; Kavakli-Loucopolos 2004]. So, this section focus on the *i** framework. However, some aspects of the Inquire-Cycle [Potts-et-al. 1994] and the KAOS Approach [Dardenne-et-al. 1993], are also mentioned, due to their relevance in the goal-directed acquisition process.

The Inquire-Cycle model [Potts-et-al. 1994], provides an structure for describing and supporting discussions about system requirements. The Inquiry-Cycle model has three phases:

- **Requirements documentation.** The stakeholders write down proposed requirements. For analyzing requirements interviews, technical documentation for similar systems can be applied. At this stage, scenario analysis is a valuable

technique. Scenarios can be documented by means of use cases (a short description with a number attached), *scripts* or *action tables* (tables or diagrams that identify both the action and the agent of the action). To tackle complexity, sequences of actions in scenarios can be represented at two levels of complexity: scenarios that have subcases belong to *complete scenarios* of *families of scenarios*, whilst the shared actions in the different cases are called *episodes* or *phases*.

- **Requirements discussion.** The stakeholders challenge proposes requirements by attaching typed annotations. This is done by means of three elements: *questions* about the requirements, *answers* to describe solutions and *reasons* that justify answers.
- **Requirements evolution.** The stakeholders attach change requests and, if relevant, they may be traced backwards to a discussion, which constitutes their rationale, and forward to the changed requirements once they has been acted on.

The KAOS (Knowledge Acquisition in autOmated Specification) Approach [Dardenne-et-al. 1993] focus on a goal-directed requirements acquisition task and is composed by three components:

- a **conceptual model** for acquiring and structuring requirements models, with an associated acquisition language;
- a set of **acquisition strategies** for elaborating requirements models in this framework; and
- an **acquisition assistant** to provide automated guidance in the acquisition process according to such strategies.

In KAOS, goals drive the identification of requirements to support them. In order to support such guidance, a goal taxonomy is defined. Goals are classified according to their pattern and their category. The pattern of a goal is based on the pattern of its formal definition. Five patterns can be identified: achieve, cease, maintain, avoid and optimize. These patterns have an impact on the set of possible behaviours of the system; *Achieve* and *Cease* goals generate behaviours, *Maintain* and *Avoid* goals restrict behaviours, and *Optimize* goals compare behaviours [Dardenne-et-al. 1993].
Goals can also be of different categories:

- *SystemGoals* are application-specific goals that must be achieved by the composite system. They can be further specialized in:
  SatisfactionGoals concerned with satisfying agent requests,
  InformationGoals concerned with getting agents information about object states,
  RobustnessGoals concerned with maintaining the consistency between the automated and physical parts of the composite system.
  SafetyGoals and *PrivacyGoals* concerned with maintaining agents in states which are safe and observable under restricted conditions, respectively
- *PrivateGoals* are agent specific goals that might be achieved by the composite system.

3.2. The *i** Framework

The *i** framework is an agent-oriented language defined by Eric Yu [Yu 1995] with the aim of modelling and reasoning about organizational environments and their information systems. For doing so, it offers a formal representation of goals and their

behaviours with a formal decomposition structure, allowing the consideration of non-functional requirements.

The *i** framework proposes the use of two types of models for modelling systems, each one corresponding to a different abstraction level: a Strategic Dependency (SD) model represents the intentional level and the Strategic Rationale (SR) model represents the rational level.

The central concept in *i** is the intentional actor. Organizational actors are viewed as having intentional properties such as goals, beliefs, abilities, and commitments. Actors depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. By depending on others, an actor may be able to achieve goals that are difficult or impossible to achieve on its own. On the other hand, an actor becomes vulnerable if the depended-on actors do not deliver. Actors are strategic in the sense that they are concerned about opportunities and vulnerabilities, and seek rearrangements of their environments that would better serve their interests.

A SD model consists of a set of nodes that represent actors and a set of dependencies that represent the relationships among them. Dependencies expresses that an actor (depender) depends on some other (dependee) in order to obtain some objective (dependum). Thus, the depender depends on the dependee to bring about a certain state in the world (goal dependency), to attain a goal in a particular way (task dependency), for the availability of a physical or informational entity (resource dependency) or to meet some non-functional requirement (softgoal dependency).

Actors can be specialized into *agents*, *roles* and *positions*. A position covers roles. The agents represent particular instances of people, machines or software within the organization and they occupy positions (and as a consequence, they play the roles covered by these positions). The actors and their specializations can be decomposed into other actors using the *is-part-of* relationship.

A SR model allows visualizing the intentional elements into the boundary of an actor in order to refine the SD model with reasoning capabilities. The dependencies of the SD model are linked to intentional elements inside the actor boundary. The elements inside the SR model are decomposed accordingly to two types of links:

- *Means-end* links establish that one or more intentional elements are the means that contribute to the achievement of an end. The "end" can be a goal, task, resource, or softgoal, whereas the "means" is usually a task. There is a relation *OR* when there are many means, which indicate the different ways to obtain the end. The possible relationships are: *Goal-Task*, *Resource-Task*, *Task-Task*, *Softgoal-Task*, *Softgoal-Softgoal* and *Goal-Goal*.
- *Contribution* links are *Means-end* links with a *softgoal* as end it is possible to specify if the contribution of the means towards the end is negative or positive.
- *Task-decomposition* links state the decomposition of a task into different intentional elements. There is a relation *AND* when a task is decomposed into more than one intentional element. It is also possible to define constraints to refine this relationship. The importance of the intentional element in the accomplishment of the task can also be marked in the same way that in dependencies of a SD model.

SR models have additional elements of reasoning such as *routines*, *rules* and *beliefs*. A *routine* represents one particular course of action (one alternative) to attain the actor's goal among all alternatives. *Rules* and *beliefs* can be considered as conditions that have to fulfil to apply routines.

The graphical notation is shown in figure 3.1 using an example about academic tutoring of students. On the left-hand side, we show the SR model of a tutor and the hierarchical relationships among their internal intentional elements. On the right-hand side, we show the SD dependencies between a student and a tutor.

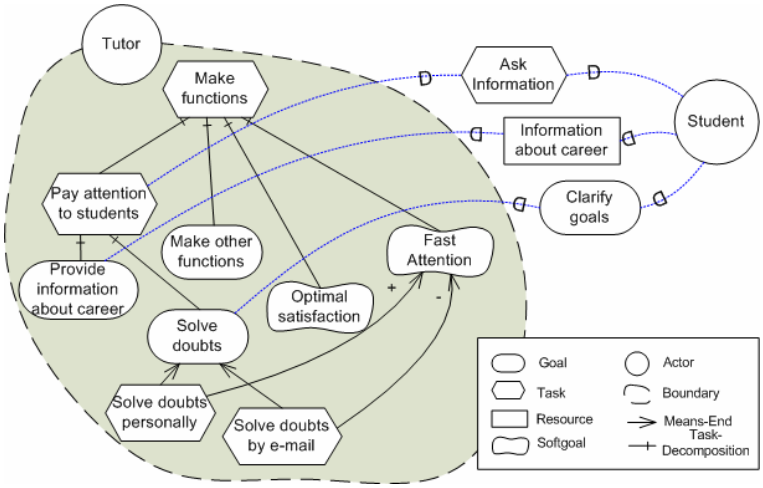

Fig. 3.1. Example of an *i** model for an academic tutoring system.

These are the concepts as described in [Yu 1995]. However, a characteristic that is soon discovered when starting to use *i** is that there is not a single definition of the language. Due to the strategic nature and objectives of *i**, the language provides some degree of freedom that results into the tendency of each research team to create its own customized *i**. That's why they are multiple variants of the language, which are identified in [Ayala-et-al. 2005]. Some of the *i** variants in process of consolidation are the Goal-oriented Requirement Language (GRL) [Amyot-Mussbacher 2002; GRL-web] and the language of the TROPOS method [Bresciani-et-al. 2004; Tropos-web]. In [Ayala-et-al. 2005] there is a definition of a reference framework, to be used in the analysis and classification of the analyzed *i** variants.

3.3. *i** Methodologies

The *i** framework is used in a wide variety of context. As key factors of its success are the visual utility and the reasoning capabilities it provides, less attention have been given to the construction of the models themselves. For instance, in the seminal proposal of the *i** framework [Yu 1995], the construction of the models is not addressed as an issue on its own and only small guidance is provided (mainly the construction of SR models by using strategic reasoning in order to obtain the SD dependencies). Most of the proposals on *i** already assume that the user already has a technique to build the models which is not always true.

The organizational patterns provided in [Kolp-et-al. 2001; Kolp-et-al. 2003] are based on organizational structures and can be used as a basis to generate *i** models. Despite

this approach is suitable when a specific organizational architecture is modelled, there is no guidance on how to use the patterns and no description on how this patterns are defined. Thus, as far as we know, the only detailed approaches on constructing *i** models are:

- the Tropos methodology [Bresciani-et-al. 2004] which is intended to support all analysis and design activities in the software development process;
- the *Goal-based Business Modelling oriented towards late requirements generation* method [Estrada-et-al. 2003];
- a methodology for building *i** models based on activities theory [Neto-et-al. 2004];
- the RESCUE process [Jones-Maiden 2004].

The Tropos methodology [Bresciani-et-al. 2004] is intended to support all analysis and design activities in the software development process. Tropos model information systems as social structures by means of a collection of social actors, human or software, which act as agents, positions, or roles and have social dependencies among them.
The models produced by Tropos are based on the concepts of actor, goal, plan, resource, dependency, capability and belief (see the language metamodel described in [Bresciani-et-al. 2004], section 5). These concepts are associated to different modelling activities that contribute to the requirements acquisition and its refinement and evolution into subsequent models such as actor modelling, dependency modelling, goal modelling, plan modelling and capability modelling.
In Tropos the requirements analysis is split in two phases: Early Requirements and Late Requirements analysis, both sharing the same conceptual and methodological approach. Thus, the five main development phases of the Tropos methodology are the following:

- Early Requirements
  In that phase, the domain stakeholders are identified and modelled as social actors. These actors depend on one another for goals to be achieved, plans to be performed, and resources to be furnished. These dependencies between actors allow to state the why behind the system functionalities and, as a last result, to verify how the final implementation matches initial needs.
- Late Requirements
  The conceptual model is extended including the system as a new actor, and its dependencies with the other actors of the environment. These dependencies define all the functional and non-functional requirements of the system-to-be.
- Architectural Design
  The system's global architecture is defined in terms of sub-systems (represented as actors), interconnected through data and control flows (represented as dependencies). A mapping of the system actors to a set of software agents, each characterized by specific capabilities, is also provided.
- Detailed Design
  Agent capabilities and interactions are specified. If the implementation platform has already been chosen, it can be considered in order to perform a detailed design that will map directly to the code.
- Implementation
  The detailed design specification is used as a basis of the established mapping between the implementation platform constructs and the detailed design notions.

The objective of the *Goal-based Business Modelling oriented towards late requirements generation* method [Estrada-et-al. 2003] is to use a business model for constructing a

software requirements specification. The method uses a Goal-Based Elicitation Method in order to capture the organizational context in a Goal-Refinement Tree. This tree is the basis to create *i** SD models, which can be used to perform business improvement analysis. Finally, strategic models can be derive functional (use case) specifications with their corresponding scenarios.

The methodology presented in [Neto-et-al. 2004], proposes to build *i** models based on activities theory. Taking the activity theory models as a starting point, the activities and their actions are analysed to construct the model. The method provides concrete guidelines for mapping this concepts to an SD *i** models, focusing on the goals**.** SD models are used to build the SR. This methodology covers Tropos Early Requirements and Late Requirements analysis.

The RESCUE process [Jones-Maiden 2004] is applied in the requirements specification stage of the project and uses four different techniques that are mutually supportive:

- human activity modelling is used to analysis the current work domain;
- *i** goal modelling is used to perform the system goal modelling;
- use case modelling and specification is done by applying systematic scenario; walkthroughs and scenario-driven impact analyses; finally,
- requirements management is done.

The RESCUE process provides some guidelines for constructing the *i** model, however, it is aimed at discovering/eliciting requirements through a bi-directional coupling of the *i** model elements and the use cases and scenarios, allowing movement from one model to the other, and viceversa.

The method proposed in [Santander-Castro 2002] also addresses the construction of a use case specification from an *i** model. However, it cannot be considered an *i** methodology, because it assumes that the models are already build correctly before its execution. Thus, its goal is to guide the mapping and the integration process of *i** organizational models and Use Cases. For doing so it provides some guidelines that allows to discover the actors and the use cases for the actors from the SD model, and the scenarios of the use cases from the SR model.

Table 3.6 summarizes the aspects addressed by the different methodologies.

Table 3.6. Summary of the issues addressed by the *i** methodologies.

Methodology	Developed models	Technique use for context analysis	Phases of the lifecycle addressed	Main output
TROPOS [Bresciani-et-al. 2004]	SD, SR	Goal-Based Elicitation Methods	From Early Requirements to Implementation	Support for all analysis and design activities
[Estrada-et-al. 2003]	SD, SR	Goal-Based Elicitation Methods	Requirements and Specification	*i** model to be used in Specification
[Neto-et-al. 2004]	SD, SR	Activity Theory	Early Requirements Late Requirements	*i** model to be used in Specification
RESCUE [Jones-Maiden 2004]	SD, SR	Human Activity Models	Requirements	Use Case specification
[Santander-Castro 2002]	-	Already build *i** SD and SR model	Specification	Use Cases Specification

3.4. Some *i** and Goal-Oriented Evaluation Techniques

The evaluation of *i** models is addressed in the literature. The systematic evaluation of process alternatives is already addressed in Yu's work [Yu 1995]: the SD model supports the systematic identification of stakeholders and their interests and concerns, whilst the SR model supports the systematic evaluation of alternatives through the concepts of ability, workability, viability, and believability. The Tropos project [Bresciani-et-al. 2004] uses the *i** capabilities in a similar way in order to connect strategic reasoning with information system development.

As *i** is a goal-oriented technique, some goal-oriented analysis methods, such as the NFR framework [Chung-Nixon-Yu 2000] and the AGORA method [Kaiya-et-al. 2002], can be applied.

The NFR framework [Chung-Nixon-Yu 2000] uses non-functional requirements to drive design, to support architectural design, and to deal with change. It is based on making explicit the relationships between quality requirements and design decisions, which it is done by applying the following tasks: 1) Develop the NFR goals and their decomposition; 2) Develop architectural alternatives; 3) Develop design tradeoffs and rationale; 4) Develop goal criticalities; and 5) Evaluation and Selection.

The AGORA method [Kaiya-et-al. 2002] provides a techniques for estimating the quality of requirements specifications in a goal-oriented setting. Its execution is top-down and supports the following aspects: selection of the goals to be decomposed; priorization and stakeholders goal conflict solving, selection of a goal out of the alternatives of the goals as a requirements specification; and analysis of the impacts when requirements change.

The structural analysis of actor-dependency models is addressed in the REACT method [Franch-Maiden 2003] by defining metrics over the models with respect some properties considered of interest for the modelled system (such as security, accuracy or efficiency). More details about this method are provided in section 1 and 5.

3.5. Applications of the *i** Framework

In its thesis, Eric's Yu [Yu 1995] proposes to apply the *i** framework in the context of requirements engineering, business process reengineering, organizational impacts analysis and software process modelling. This broad scope gives rise to an extended practice on the construction of *i** models.

In [*i**-web] Eric Yu has compiled all the publications on *i**. As the community is growing a more collaborative web page has been designed in the form of a wiki [*i**-wiki]. In the last, people on the community can add publications, case studies and describe the *i** tools they have develop. The publications on *i** are related to the following fields:

- **Requirements engineering.** The work on this area are related on how to use *i** in order to perform requirements engineering for system development. Related to that category but focusing on security aspects and the human-user interface, some other

work is undertaken on *security requirements engineering* and *variability and personalization*.

- **Business Modelling.** The work in this area explores how to model business in the *i** framework*, software engineering processes and organizations* and *systems and organizational architecture* can also be model and several approaches can be applied to perform *process analysis and design* and *reengineering* activities. The REACT method [Franch-Maiden 2003] and the organizational analysis proposed by [Kolp-et-al. 2001; Kolp-et-al. 2003; Bastos-Castro 2003] fall into this kind of work.
- **Agent-Oriented Systems Development.** The agent-oriented community uses the *i** approach to assess agent-based systems in all its development phases. A widespread example of this work is the Tropos project [Bresciani-et-al. 2004; Tropos-web]. The reasoning capabilities provided by *i* evaluation, verification and validation* of the resulting systems and allows to explore general aspects on *trust* in Multi-Agent Systems.
- **Management issues.** The *i** framework is also used to investigate on *data management processes*, knowledge *management* and *intellectual property management*.

The link between strategic reasoning and information system development has been widely addressed [Bresciani-et-al. 2004; Jones-Maiden 2004; Santander-Castro 2002]. This proposals provide guidelines for mapping an *i** model to an UML use cases and classes specification, among them we remark [Santander-Castro 2002].

3.6. Remarks

The *i** framework is becoming a consolidated requirements engineering technique. However, its use stills having some open issues such as the ambiguity of the modelling language, the lack of prescriptive methodologies or the big size of the resulting models.

Despite of this, the *i** framework as been successfully used in a wide variety of context such as requirements engineering, business modelling, process analysis, agent-oriented system development, process management or process reengineering. From the *i** publications, we can observe that every research group has adapted the framework for its own purposes (i.e. by adding new model constructors and semantics) and defined its own methods for constructing the models. However, the main concepts of the framework remained unchanged.

On this basis, the *i** framework is suitable for modeling and evaluating COTS architectures. Although, traditional architectural techniques model architectures at the technical level, the process level provided by *i** links requirements and architectures and allows a better communication with stakeholders. The evaluation of goal-oriented approaches, using functional and non-functional techniques has already been addressed. Actually, the use of structural metrics for evaluating the model architectures has been applied in the REACT method [Franch-Maiden 2003].

As REACT is an on-going work, it has not addressed yet the lack of a methodology. The agent-oriented paradigm has many points in common with the *i** framework (mainly the use of intentional agents) and that's why this discipline is analyzed in the next section.

4. The Agent-Oriented Paradigm

4.1. Introduction to the Agent-Oriented Paradigm

The origin of agents is located in the artificial intelligence discipline. At the beginning, the concept of agents was used to refer to special programs without a detailed nature and implementation. However, since the 1980s, the research in software agents and multi-agent systems has increased, not only in the artificial intelligence field, but also in other computing areas such as distributed computing, object-oriented systems, software engineering, economics, sociology, and organisational science. In order to understand the success of agents in those fields, basic concepts definitions are needed.

The knowledge level hypothesis [Newell 1982] states that there exists a distinct computer level, characterized by having the knowledge as the medium and the principle of rationality as the law of behaviour. At this level, the system is an agent that processes its knowledge to determine the actions to take for attaining its goals. Although there is no official definition of what constitutes an agent, the following characterization is increasingly being used:

> *"An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives"*
>
> Michael Wooldridge (1997) [Wooldridge 1997]

In the knowledge level, agents are considered alone, without taking into account interactions between them. The social level hypothesis [Newell 1982] states that there exists a computer level immediately above the knowledge level, called the social level, which is concerned with the inherently social aspects of multiple agent systems. In [Jennings-Campos 1997] this social level is empathised as it provides an abstract characterisation of those aspects of multi-agent system behaviour that are inherently social in nature such as those concerned with representing phenomena such as co-operation, co-ordination, conflicts and competition.

When the key abstraction of a system is an agent, we have *agent-based systems*. Although agent-based systems may have only a single agent, many problems involve multiple agents. Thus, Multi-Agent Systems (also called MAS) exhibit a greater potential and present the following characteristics [Jennings 1998]**:**

- each agent has incomplete information, or capabilities for solving the problem, thus each agent has a limited viewpoint;
- there is no global system control;
- data is decentralized; and
- computation is asynchronous.

Because of that, the agents involved exhibits properties such of autonomy, reactivity, pro-activeness and social ability [Wooldridge-et-al. 2000], and Multi-Agent Systems

contributes to an increased processing speed-up, reduced communication bandwidth, increased reliability [O'Malley-DeLoach 2001].

This conceptual framework is based on the concept of autonomous agents interacting with one another for their individual and/or collective good and so, it offers a natural and powerful means of analysing, designing, and implementing a diverse range of software solutions in a range of closing related disciplines [Jennings 1998]. This gives leads to the agent-oriented software paradigm, and gives rise to a considerable amount of applications in the form of frameworks and methodologies.

4.2. Agent-Oriented Software Engineering

The difficulty involved in the design and development of a software system increases with the complexity of such a system. To solve this problem, several software engineering paradigms are proposed, each one claiming to solve more problems than the one before. Despite of this, researchers continually look for new software engineering techniques [Jennings-Wooldridge 99] to improve the process.

Agents and multi-agents systems are seen as an important new direction in software engineering for the following reasons [Wooldridge-et-al. 2000]:

- **Natural metaphor.** Software participants in scenario transactions can be view as (semi-autonomous) agents.
- **Distribution of data or control.** In many software systems, the overall control of the systems is distributed across a number computing nodes that must be capable of autonomously interacting with each other. These nodes can be agents.
- **Legacy systems.** A natural way of incorporating legacy systems into modern distributed information systems is to "wrap" them with an agent layer, which will enable them to interact with other agents.
- **Open Systems.** To operate effectively in the systems where it is impossible to know all the components and interactions at design time (open systems), the ability to engage in flexible autonomous decision-making is critical.

In the development of complex and distributed systems, contemporary methods tend to fail because the interactions between the various computational entities are too rigorous defined; and there are insufficient mechanisms available for representing the system's inherent organizational structure [Jennings 2000]. The agent-oriented paradigm solves these aspects because, as mentioned in [Jennings-Wooldridge 99; Jennings 2000]:

- Agent-oriented decompositions are an effective way of partitioning the problem space of a complex system,
- The key abstractions of the agent-oriented mindset are a natural means of modelling complex systems.
- The agent-oriented philosophy for dealing with organisational relationships is appropriate for complex systems.

The appropriateness of the agent-oriented paradigm can be compared to other software engineering practices. Specifically:

- **Knowledge Engineering.** Knowledge-based systems are developed using knowledge engineering methodologies that allow to model agents cognitive characteristics. Although these techniques address multi-agents systems main

concerns (knowledge acquisition, modelling and reuse) they do not address the distributed or social aspects of the agents, or their reflective and goal-oriented attitudes [Iglesias-et-al 1999].

- **Object-Oriented Paradigm.** Objects are defined as computational entities that encapsulate some state, are able to perform actions, or methods on this state, and communicate by message passing [Wooldridge-et-al. 2000]. Both object-oriented and agent-oriented views of the system emphasise the importance of interactions between entities, thus some agent-oriented methodologies come from the object-oriented field. However, as pointed out in [Wooldridge-et-al. 2000; Iglesias-et-al 1999] there are several distinctions between the two, mainly:

  Agents embody stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agent;

  Agents are capable of flexible (reactive, pro-active, social) behaviour, whether the standard object model has nothing to say about such types of behaviour;

  A multi-agent system is inherently multi-threaded, in that each agent is assumed to have at least one thread of control.

- **Component-Based Software Engineering.** Components are self-contained computational entities, which functionality is described, encapsulated and requested. Both components and agents can be considered as a single unit of deployment because they are typically self-contained computational entities that do not need to be deployed along with other components in order to realise the services they provide, and they are able to respond requests for information about the services they provide. However, components are not autonomous in the way that we understand agents to be and there is no corresponding notion of reactive, proactive, or social behaviour in component software.

However, while agent technology represents a potentially novel and important new way of conceptualising and implementing software, it is important to understand its limitations [Jennings-Wooldridge 99]. The agent-oriented approach implies the use of software, and thus it presents the same fundamental limitations as more conventional software solutions. For instance, agent technology is limited by the capabilities of artificial intelligence techniques.

4.3. Agent-Oriented Methodologies

The success in the development of agent and multi-agent systems is often conditioned to the use of a methodology. According to [Sturm-Shehory 2003], a methodology is the set of guidelines for covering the whole lifecycle of system development both technically and managerially.

Agent-oriented methodologies are composed by the analysis, design, and implementation phases, which are done at a level of abstraction more adequate to the problem to be solved. According to [Burmeister 1996], during the analysis phase the acting entities of the problem domain are identified and modelled as agents. Agents and their actions (or behaviour) are refined and specified in the design phase. Finally, at the implementation phase, agents are programmed with the aid of an agent-oriented programming language or using a multi-agent development environment. In [Jennings-

Wooldridge 99] an agent-oriented software life-cycle also proposes a verification phase, where verification can be done either by using axiomatic approaches or semantic approaches.

There are many agent-oriented methodologies available and none of them is suitable for all purposes. As pointed out in [Sturm-Shehory 2003] this has implications in the following fields:

- **Industrial problem:** industrial developers must select one of the methodologies, which can become a non-trivial task.
- **Standard problem:** with no standard available, potential industrial adopters of agent technology refrain form using it.
- **Research problem:** excessive efforts are spent on developing agent-oriented methodologies, sometimes producing overlapping results.

Because of that, the analysis and comparison of agent-oriented methodologies has been widely studied [Iglesias-et-al 1999; Wooldridge-et-al. 2000; Sturm-Shehory 2003; Tveit 2001; Cernuzzi-Rossi 2002; Sabas-et-al. 02; Dam-Wnikoff 2003; Sturm-Shehory 2003; Yu-Cysneiros 2002; Sudeikat-et-al. 2004].

Agent-Oriented methodologies are generally built by extending other existing methodologies, mainly object-oriented methodologies and knowledge engineering methodologies, in order to include the relevant aspects of the agents [Iglesias-et-al 1999]. Some other methodologies follow formal approaches or come from the scratch. Figure 3.2, shows a genealogy of agent-oriented methodologies. This genealogy is an excerpt of the one proposed in [Sudeikat-et-al. 2004], containing only the most cited methodologies.

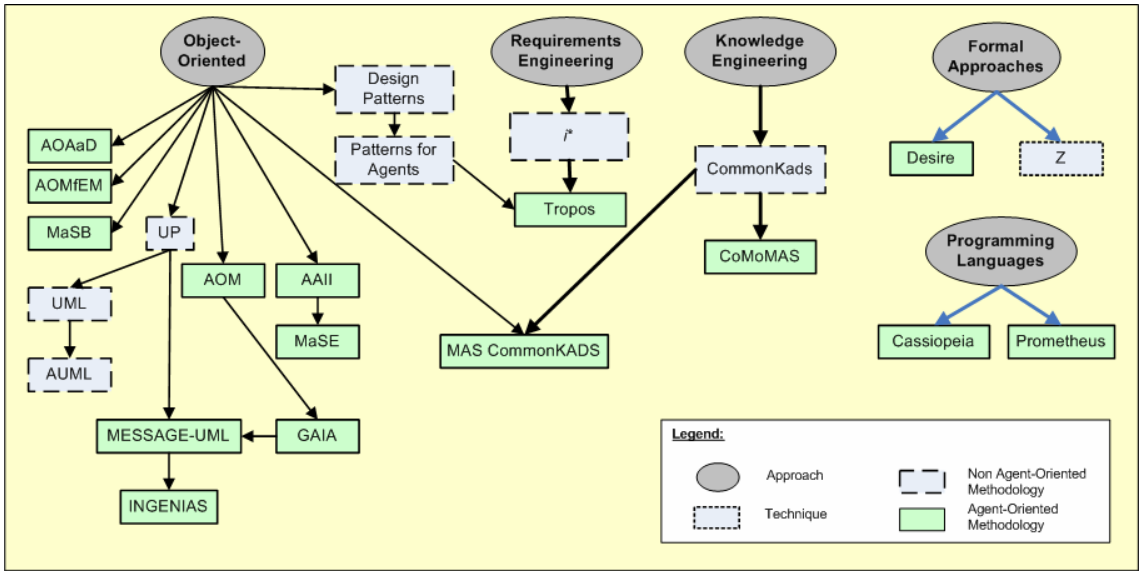

Fig. 3.2. Excerpt of [Sudeikat-et-al. 2004] agent-oriented genealogy.

An alternative classification of the methodologies is presented in [Wooldridge-et-al. 2000] depending on how the models are constructed:

- Top-down approaches, which are based on progressive decomposition of behaviour, for instance the Agent Modelling Techniques for Systems of BDI Agents [Kinny-et-al. 2006] employs an iterative top-down approach to develop its models.
- Bottom-up approaches, which begin by identifying elementary agent behaviours such as Cassiopeia [Collinot-et-al. 1996].

- Mixed approaches, such as Gaia [Wooldridge-et-al. 2000] employs fine-grained models (bottom-up obtained) and more generic models (top-down obtained) to capture the result of the analysis and design process.

Due to the vast amount of agent-oriented methodologies, a detailed description of each of them is not be given. Instead, only representative methodologies on each category are presented: the CommonKADS as a knowledge engineering methodology, the GAIA Methodology as an extended object methodology, and CASSIOPEIA as coming from other methodologies. Due to the relevance of the TROPOS methodology in the *i** framework, it is detailed in section 3.3.

Analysis and Design of multiagent systems using MAS-CommonKADS

The MAS-CommonKADS methodology [Iglesias-et-al. 1998] extends the knowledge engineering methodology CommonKADS [Schreiber-et-al. 1994] by adding techniques from object-oriented methodologies and from protocol engineering for describing the agent protocols. In this methodology, the analysis and design of the system are made by developing several models. For each model, the methodology provides a textual template for describing the entities to be modelled (named constituents) and a set of activities for building every model, based on the development state of every constituent (empty, identified, described or validated).
The models defined in the phases of the methodology, which are:

1. Conceptualization Phase. In this first informal phase, user requirements are collected by following a user-centred approach that determines some use cases (scenarios) in order to understand informal requirements and to test the system.
2. Analysis Phase. In the analysis phase the following steps are undertaken:
   Agent modelling describes the main characteristics of the agents, including reasoning capabilities, skills (sensors/effectors), services, or goals.
   Task modelling describes agent's goals, tasks and task decomposition by using textual templates and diagrams.
   Coordination modelling describes interactions, protocols and required capabilities of the conversations between agents.
   Knowledge modelling describes both the agents knowledge and the environment knowledge.
   Organisation model describes both the organisation in which the MAS is going to be introduced and the organisation of the agent society
3. Design Phase. During the analysis, the following models are developed based on the previous models:
   Agent design composes or decomposes the agents according to pragmatic criteria and selection of the most suitable agent architecture for each agent.
   Agent network design determines the relevant aspects of the infrastructure of the MAS-System.
   platform design consist of the selection of the software and hardware that is needed or available for the system.

The Gaia Methodology for Agent-Oriented Analysis and Design

GAIA [Wooldridge-et-al. 2000] is an agent-based methodology intended to provide a systematic way to go from a statement of requirements to a detailed design that could be implemented directly. The methodology proposes the development of a set of models, which move from abstract to increasingly concrete concepts (see figure 3.3):

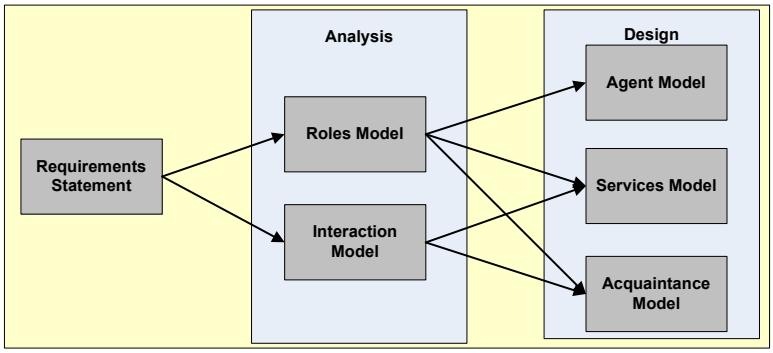

Fig. 3.3. Relationships between the GAIAs Models.

1. Requirements Statement. Independent of the paradigm used for analysis and design.
2. Analysis Stage. During the analysis stage, the roles of the system are identified and documented in the *roles model*, whilst the identification of the recurring patterns of interaction that occur in the system between the various roles are represented in the *interaction model*.
3. The Design Stage. The abstract models derived during the analysis phase are used to build to more concrete models. Thus, in the *agent model*, roles are aggregated into agent types, and refined to form an agent type hierarchy. The *service model* identifies the services associated to each agent and its main properties. Finally, the interaction model and agent model are used to develop the *acquaintance model*, which is a directed graph that defines the communication links between kinds of agent.

CASSIOPEIA: a Methodology for Agent Oriented Design.

Cassiopeia [Collinot-et-al. 1996] is a methodological framework that has essentially been developed for the design of systems where collective behaviours are put into operation through a set of agents. Cassiopeia proposes to design MAS in terms of agents; which are provided with three levels of behaviour: elementary, relational, organizational. Due to the dynamic behaviour of the agents, the designer only takes into account the organizational structures between agents, which will be instantiated within the problem-solving context. Cassiopeia is considered as a bottom-up approach [Wooldridge-et-al. 2000] as it proposes to start from the *behaviours* required to carry out some task and performing three steps that reconcile both local and global views (figure 3.4):

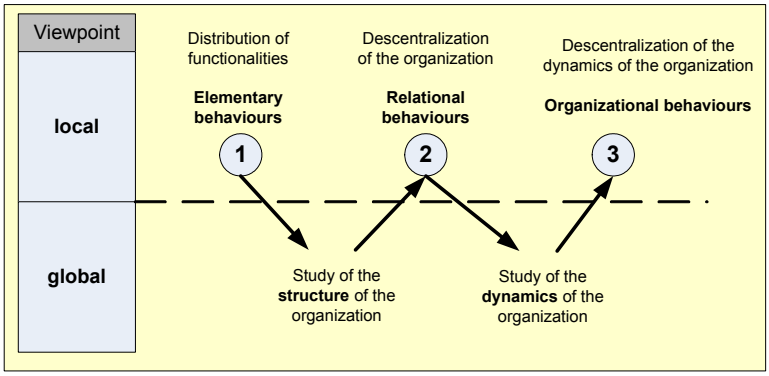

Fig. 3.4. The Cassiopeia method overview.

Following this approach, the methodology proposes three steps:

1. Identify the *elementary behaviours.* The different types of agents are defined based on the elementary behaviours that are required for the achievement of a considered collective task. These behaviours are identified on a previous functional or object-oriented step.
2. Identify *the relational behaviour*. The organization structure is analysed by means of the dependencies between the elementary behaviours identified in the previous step. This dependencies are represented in a coupling graph that is analysed in order to (a) remove inconsistent dependencies; (b) ignore dependencies according to the available heuristics of the application domain. The result is the influence graph, which contains only those dependencies supposed relevant to the achievement of the task; and can be used to analyse how agents identify and handle different elementary behaviours.
3. Identify *the organisational behaviours*. The behaviours that will enable the agents' management of formation, durability and dissolution of groups are specified. A description of the organization dynamics is obtained by (a) identifying the trigger agents by using the influence graph; (2) determine, for each of them, the selection methods allowing controlling the formation of groups.

4.4. Comparing and Evaluating Agent-Oriented Methodologies

The relevance and growing importance of the agent-oriented methodologies, results into a big amount of literature on comparing and evaluating agent-oriented methodologies. For instance, [Iglesias-et-al 1999] provides a classification of methodologies, and describes and compare them. In [Sabas-et-al. 02; Dam-Wnikoff 2003; Sturm-Shehory 2003; Sudeikat-et-al. 2004] several methodologies are evaluated and compared according to a set of established criteria, different in each case. [Cernuzzi-Rossi 2002] proposes to compare methodologies by using an attribute tree and a questionnaire that is filled for each methodology. An evaluation formula is then provided to obtain a numerical evaluation of the methodology according to the information in the questionnaire. Finally, [Yu-Cysneiros 2002] proposes the use of an exemplar for evaluating the methodologies, providing a proposal of which could be these exemplar.

4.5. Remarks

As it is mentioned in these section, both components and agents gave similarities in the way that they are self-contained computational entities deployed independently, but able to provide some services. Despite components are not as autonomous as agents, they interact with other components to achieve or provide certain functionalities as agents do. On the other hand, humans that interact with the components exhibit the characteristics of agents (just because agents are inspired in human knowledge and behaviour). Thus, most of the concepts identified in the agent-oriented methodologies such as roles and agents, or interaction mechanisms, can also be applied to a COTS-Based System.

5. Business Process Reengineering

5.1. What is Business Process Reengineering?

The term business process reengineering has been used in the business milieu since the beginning of the XX century, but is in the 1990's with the inclusion of the new information technologies that it begins to get applied consciously in the companies. The reengineering of a business process consists in changing that process in order to improve its competitiveness. This improvement can be performed by adding an information system to the process or making some organizational changes such as incorporating outsourcing of services, changing its structure by empowerment, or applying just-in-time strategies.

The definition of what involves business process reengineering and in what contexts it applies, is not clearly defined, and often the terms business process redesign and business process reengineering are used indistinctly without clearly stating the difference between them. To understand the scope of business process reengineering it is necessary to analyse its definition:

> *The fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service and speed.*
>
> Michael Hammer and James Champy [Hammer-Champy 93]

Although this definition is very accepted and used in the community, some authors disagree with some of the terms. On the one hand, some authors disagree with the fact that the improvements on the process have to be dramatic and radical. For instance, [Jarzabek-Tok 1996] argues that while some companies require radical re-thinking of business process others may benefit from improvement or innovation; and [Katzenstein-Lerch 2000] considers that the simple fact of installing an information system, explicitly or implicitly redesigns the organizational process in which it is embedded. On the other hand, some authors claim a broader context for redesign, for example [Grant 2002] states that focusing only on the business process, other important aspects of the organizations are ignored (e.g. organizational structure, people, communication, or technology).

Traditionally, the major activities in a typical reengineering effort would include [Yu 1995]:

- Identify, delineating, and modelling the existing process.
- Analyzing it for deficiencies.
- Proposing new solutions (process design).
- Implementing the new design, in terms of new technical systems and new organizational (people) structures (roles and responsibilities).

There are also controversies about modelling the current process in order to analyse its deficiencies. Michael Hammer and James Champy, for instance, propose the philosophy of beginning from a 'clean slate' [Hammer-Champy 93], which defends the redesign of the new process without taking into account the existing one. In [Davenport-Stoddard 1994] this point is addressed as *the myth of the clean slate*, with the argument that beginning from the scratch is not always possible, because it is difficult and expensive to replace the equipments, technology, people skills, and knowledge that an organization currently has. Thus, the myth of the *clean slate* differentiates *clean slate* design than *clean slate* implementation in order to get the innovation benefits of designing from the scratch, but implement them taking into account the existing resources.

Actually, there is a positive correlation between dramatical changes and perceived success [Teng-et-al. 98]. However, many authors and practitioners argue for the previous analysis of the existing process and the generation of alternatives from that analysis [Jarzabek-Tok 1996; Katzenstein-Lerch 2000; Anton-et-al. 1994; Giaglis 2001; Grant 2002]. According to [Anton-et-al. 1994], a previous analysis allows the identification of local inefficiencies in business processes, and recommends interventions to remove or mitigate them.

Taking these different aspects into account, the term of process reengineering is used from its wider point of view in order to consider both radical and minor changes in process redesign; and both the possibility of studying the current process or beginning from a clean slate.

5.2. Modelling Business Process Reengineering Processes

Organisations are complex in nature because they usually involve different people, business units, resources and systems which interact via many different processes. Consequently, carefully developed models are necessary for understanding their behaviour in order to design new systems or improve the operation of existing ones [Giagli 2001]. Because business process reengineering can be applied from different disciplines, it is possible to use many different modelling tools and languages to represent the process. However, not all the existent models cover the same aspects, and the choice of a business process modelling technique depends on the discipline viewpoint adopted.

There are several studies that prove that the representation of the problem affects its resolution [Katzenstein-Lerch 2000]. However, in spite of the large number of process representations available for process redesign, no criteria exist for what constitutes a good process model for redesign. There are, indeed, some classification criteria (e.g., [Curtis-et-al. 1992; Eertink-et-al. 1999; Katzenstein-Lerch 2000; Giaglis 2001) and evaluation frameworks that can support this task (e.g., [Curtis-et-al. 1992; Janssen-et-al. 1997; Katzenstein-Lerch 2000]).

In the context of this work, the most appropriate of these frameworks is the one proposed by [Katzenstein-Lerch 2000], which establishes the following classification of modelling languages:

- **Traditional systems techniques**: commonly used for developing information systems, these models capture process activities and the flow of items and information (e.g., Flowchart and Data flow diagramming, IDEF1x techniques and UML)
- **Coordination models:** provide a generally organizational view and highlight what conditions (for instance, resources, circumstances or states of the system) must be met at each stage in a process before the process can continue (e.g., Role interaction nets, role activity diagram)
- **Sociotechnical qualitative systems**: provide many useful concepts such as personal and organizational goals, variances, rich information, and causal and structural relationships (e.g., ETHICS, Soft Systems Methodology, Multiview and *i** models).

The classified languages are evaluated in [Katzenstein-Lerch 2000], according to the following evaluation criteria:

- **Content criterion.** Evaluates how the languages capture the social context, mainly the modelling elements and its nature (logistic, psychological or sociological).
- **Process status criteria.** Evaluates how the language captures the current state of the process: process emergence, multiple operations routines and missing resources or information flows.
- **Presentation and use criteria.** Involves the ability needed to use the model's elements productively, including the representation of the elements (visual utility), the support for qualitative reasoning and the heuristics for process redesign.

Several languages are evaluated in [Katzenstein-Lerch 2000] following these criteria, table 3.7 presents the results obtained.

Table 3.7. Features of Process Models. Obtained from [Katzenstein-Lerch 2000].

	IDEF	Dataflow Diagrams	Role Activity Diagrams	Rummler -Brache	Action Work-flows	*i**	Ethics	Multi-view	GED frame-work
I. Process Content Criteria									
1. Capture social context	-	-	-	-	~	~	✓	✓	✓
II. Process Status Criteria									
2. Capture exceptions	-	-	-	-	~	-	✓	-	✓
2a. Multiple Operational Routines	-	-	-	-	-	~	-	✓	✓
2b) Unfulfilled Dependencies	-	-	-	-	✓	-	~	-	✓
III. Presentation and Use Criteria									
3. Visual Utility	-	-	-	-	-	-	-	✓	✓
4. Qualitative Reasoning	-	-	-	-	-	✓	-	✓	✓
5. Heuristics for Redesign	-	-	-	~	-	-	✓	✓	✓
Model Elements									
Roles	✓	-	✓	✓	✓	✓	✓	✓	✓
Goals	-	-	~	~	~	✓	✓	✓	✓
Dependencies	✓	✓	✓	✓	✓	✓	✓	✓	✓
Seqüencial Flow	✓	✓	✓	✓	✓	-	-	✓	-

(✓) addresses the issue fully (~) deals with the issue but not fully (-) does not deal with the issue

5.3. Methodologies for Business Process Reengineering

The use of methodologies to guide Business Process Reengineering is a common fact. But, as most of them are used in consulting firms (Andersen consulting, CSC/Index, Ernst and Young, McKinsey, SRI), many of them are not available do to intellectual property rights.
There is no formal classification for process reengineering methodologies but it is possible to identify those that begin from a *clean slate* [Hammer-Champy 93] in order to plan a radical change; from those that take the existing process as a departing point for a simpler redesign. It is also possible to distinguish those methodologies that explicitly use goal-oriented approaches [Lee 1993; Anton-et-al. 1994; Grover-Malhotra 1997; Jarzabek-Tok 1996 ; Yu-et-al. 1996; Brynjolfsson-et-al. 1997; Kueng-Kawalek 97; Katzenstein-Lerch 2000; Koubarakis-Plexousakis 2000] from those that do not [Kettinger-et-al. 1993; Grover-et-al 1994; Klein 1994].

Most of the proposed methodologies are designed to focus on a particular kind of problem and usually enclose their own business process modelling language. Hence, we have different methodologies and modelling languages depending on the reengineering aspect that is focused. As it is shown latter on, some methodologies focus on how to take the strategic goals of the organization into account while others centre on how to explore different alternatives for the process or how to evaluate them.

Figure 3.5 shows a generic overview of all possible stages in a BPR process. The stages and activities have been obtained from [Teng-et-al. 98].

Stage 1. Identification of BR Opportunities
- Establish a steering committee for overall BR planning
- Secure management commitment
- Align with corporate and IT strategies
- Identify major business processes with a "business model"
- Understand customers' requirements
- Prioritize processes and select one for implementation

Stage 2. Project Preparation
- Plan for organizational change (e.g., inform stakeholders)
- Organize a BR team for the selected process
- Train the BR team members
- Conduct project planning

Stage 3. Analysis of the Existing Process
- Analyze existing process structure and flows
- Identify value-adding activities
- Identify opportunities for process improvement

Stage 4. Development of a Process Vision
- Understand process customers' requirements
- Identify process performance measures
- Set process performance goal
- Identify IT that enables process redesign
- Develop a vision (preliminary sketch) for the redesigned process

Stage 5a. Solution: Technical Design
- Develop and evaluate alternative process designs
- Detailed process modeling (entities, relationships, etc.)
- Design controls for process integrity
- IS analysis and design for the new process
- Prototype and refine the process design

Stage 5b. Solution: Social Design
- Empower customer contact personnel
- Define jobs and incentives
- Develop and foster shared values
- Define skill requirements and career paths
- Design new organizational structure
- Design employee performance measurement schemes
- Design change management program

Stage 6. Process Transformation
- Develop test and rollout plans
- Implement the social and technical design
- Train staff and pilot new process

Stage 7. Process Evaluation
- Monitor performance
- Continuous improvement

Fig. 3.5. Stages and tasks from rengineering projects. From [Teng-et-al. 98].

These stages can be considered generic, as the different methodological approaches usually undertake different stages depending on their particular orientation [Grover-

Malhotra 1997]. Hence, some business process reengineering methodologies focus on obtaining the strategic goals, whilst some others may focus on the generation or the evaluation of alternatives. Some of these specific methodologies are described in the following sections, grouped by its focus:

- goal-oriented strategy acquisition;
- generation and evaluation of alternatives; and
- formal specification and execution process.

Some other methodologies, tools and techniques, following a more general approach, can be found in [Klein 1994; Kettinger-et-al. 1993; Kettinger-et-al. 1997; Kueng-Kawalek 97; Grover-Malhotra 1997].

5.3.1. Goal-Oriented Business Process Reengineering Methodologies

A process consist as a set of sub-processes and each sub-process exists for the purpose of satisfying some functions or goals [Lee 1993], consequently an enterprise has a goal structure that has to be compatible with its physical structure in order to achieve efficiency [Anton-et-al. 1994]. Based on this idea, several authors [Lee 1993; Anton-et-al. 1994; Grover-Malhotra 1997; Jarzabek-Tok 1996; Yu-et-al. 1996; Brynjolfsson-et-al. 1997; Kueng-Kawalek 97; Katzenstein-Lerch 2000; Koubarakis-Plexousakis 2000] use goal-oriented process reengineering methodologies. There are some of these methods that specifically focus on the business goals acquisition.

[Lee 1993] models the process as a hierarchy of goals to achieve based in the identification of goals and its relations. The steps it proposes are:

- **Enumerating goals.** Goals are identified in data-driven process. Thus the data of the existent process is collected and converted into goals. If non-implemented goals arise, they are recorded in order to be taken into account in the new process.
- **Relating goals.** The identified goals are related in the hierarchy by using a provided algorism. This identification may partially be done during the first stage.
- **Checking completeness.** Completeness make sure that there is no missing goal and that all goals are accounted for the process.
- **Identify non-functional processes.** The organization may have some processes or sub-processes that are not used into the organization, if a certain sub-process doesn't have a goal associated with its parent goals means that either it is a non-functional process or that a goal has not been made explicit at the present level"
- **Exploring alternatives.** Finally, the goal hierarchy can be analysed and completed in order to explore and evaluate the different alternatives.

From a more information systems point of view, [Anton-et-al. 1994] obtains the goals from scenario analysis. The approach distinguishes two semantic categories of goals: descriptive and prescriptive goals. Descriptive goals appear on current process analysis as an operationalization of them, whereas descriptive goals come from strategic management and account for organizational structures and processes that should be observed. In the proposed process, a first hierarchy of goals is defined by taking a first set of prescriptive goals and analyse them using a top-down approach. More concrete goals are obtained bottom-up by acquiring the descriptive goals from the explanatory scenarios of the current process. Although the hierarchies of prescriptive and descriptive goals lack in levels of correspondence, the approach supports the identification of goals

and the determination of the scenarios that describe the activities that support the attainment of those goals.

5.3.2. Generation and evaluation of alternatives

One of the reasons why goal-oriented business process reengineering methodologies are popular is how they support the generation and evaluation of alternatives.

In [Lee 1993] new alternatives for a process can be explored by analysing the goals related with that process. As mentioned in section 3.4., one of the applications of the *i** approach [Yu-et-al. 1996] is to model the current process and to support the generation and evaluation of alternatives by means of the qualitative reasoning it provides.

The GED framework described in [Katzenstein-Lerch 2000] provides specific guidelines for analysing and redesigning process. Following a goal-oriented approach, it uses two different models in order to find the weaknesses of the process, define new strengths and generate solutions according to that. The GED framework is composed by two different models that are obtained from the existent process:

- *Goal/Exception diagrams* are used to recognize those goals that are unmet or that present conflicts, as well as to identify lose-lose situations.
- Based on the first analysis, new alternatives can be generated by means of taking those exceptions of the *Goal/Exception diagram* that help to achieve major goals, and convert them into a rule. Exceptions can also be used to questioning the need of a certain goal and then, finding an alternative. Finally, win-win exceptions may be taken as a base for generating new procedures.
- *Dependency diagrams* are analysed and problematic dependencies are redesigned by applying one of the following actions:
  - Alter the dependency by means of shifting the dependency to another agent or changing its nature.
  - Satisfy the dependency by means of providing the action or resources needed to accomplish it, one way to achieve that is to add information technology.
  - Balance the web of dependencies by means of reassigning the responsibilities between the process participants taking into account that a dependency is more enforced if there exist obligation from both parts.

For evaluating the process alternatives, the GED framework considers the impact of the new process dependencies on the goals and exceptions in the *goal/exception diagram* by means of: evaluate the influence of the new solution on achieving individual goals and process-level goals; study how the new solution alters the likelihood or frequency of any exceptions occurring; and observe how the new solution directly affects process-level goals without reasoning.

The matrix of change [Brynjolfsson-et-al. 1997] is an example of evaluation of alternatives in non goal-oriented process reengineering. This method identifies the critical interactions between processes and also allows capturing the interaction between the different practices. The matrix of change is composed by three different matrixes:

- An horizontal triangular matrix containing the current collection of organizational practices and which of those practices are complementary or competitive.
- A vertical triangular matrix containing the desired collection of organizational practices and which of those practices are complementary or competitive.

- A matrix interaction that bridge the other two matrixes in a transitional state that reveals the difficulty to pass from the current processes to the new ones.

As a result, the matrix of change systematizes the change management and selects those practices that better fulfil the business goals and its interpretation helps to advise change management in terms of feasibility, sequence of execution, location, pace and nature of change and, as stakeholder's are surveyed for the impact of change, the strategic coherence and value added aspects of the process can be also observed.

In goal-oriented process reengineering approaches the generation or evaluation of alternatives can also be performed by adapting reasoning techniques from other disciplines, such as AGORA [Kaiya-et-al. 2002] or KAOS [Dardenne-et-al. 1993] (see section 3.3.4)

5.3.3. Formal specification and execution of processes

Another goal-oriented methodology for business process design is the one proposed in [Koubarakis-Plexousakis 2000], which focuses on the production of detailed and formal specifications of business processes. This formal specification can be represented in a formal language and, then, verified in terms of correctness properties such as fulfilment of responsibilities assigned to roles and maintainability of constraints. The first steps of the proposed process are similar to those of the goal-oriented approaches: first, organizational objectives and goals are identified, goal reduction is performed by using different techniques some of them, such as KAOS [Dardenne-et-al. 1993] obtained from the requirements engineering field. In a second step, roles and responsibilities are identified and a match between them is established. The third step consists in specifying the primitive actions, the conditions to be noticed and their interaction with other roles. Thus, for each role, a ConGolog procedure is specified in order to discharge each role responsibilities in the fourth step. Finally, a formal verification is undertaken in order to check that ConGolog procedures local to each roles are sufficient for discharging its responsibilities.

Another proposal that focus on formalisation of process with analysis purposes is detailed in [Eertink-et-al. 1999]. In order to analyse certain behavioural properties, an operational semantics is defined and a stepwise simulation can be executed in their specific tool. Quantitative properties can also be evaluated by using queuing theory, graph models and hybrid models, as well as analysis of completion times, critical paths, resource utilisation and cost analysis.

5.4. Remarks

Business process reengineering is applied in different contexts and situations. Its success reside in the fact that, once a decision is taken it is difficult to evaluate the 'what if' other decisions had been taken. However, the level of failure of business process reengineering projects is high, approximately 70% [Grant 2002], and this leads to the continuous study of methodologies and processes in order to find key failure and success factors.

From the study of the methodologies we can observe that some of them propose to analyse the existing process and some of them don't. In some situations this is due to the adoption of a *clean slate* approach, but in other situations the authors argue that the average effort involved in analysing the current process is too high for the improvement it produces. This conclusion is presented in [Teng-et-al. 98], where the correlation of average effort with perceived success is stated based on the responses to a questionnaire filled by the industry.

In the results of the study we can notice that phases such as process evaluation, process transformation and social design are more correlated to perceived success, although the average effort to execute them is not as high as the invested in analysing the existing process.

In spite of this, in the context of the subject of this thesis proposal, we argue for a previous analysis of the existing process, as it facilitates shared process perspectives and knowledge among the process members. This reduces the impact of organizational change, and, in the specific case of COTS-Based Systems, it helps to understand what the end user is expecting from the system.

6. References

[Abts-et-al. 2000] Abst, C.; Boehm, B.; Clark, E.: "COCOTS: A COTS Software Integration Lifecycle Cost Model - Model Overview and Preliminary Data Collection Findings". Technical Report USC-CSE-2000-501, USC Center for Software Engineering, 2000.

[Alves-Castro 2001] Alves, C.; Castro, J.: "CRE: A Systematic Method for COTS Selection". In *Proceedings of the XV Brazilian Symposium on Software Engineering*, 2001.

[Amyot-Mussbacher 2002] D. Amyot, G. Mussbacher. "URN: Towards a New Standard for the Visual Description of Requirements". Proceedings of the Third International Workshop on Telecommunications and beyond: The Broader Applicability of SDL and MSC., Aberystwyth, UK, June 24-26, 2002. Pages: 21-37.

[Anton-et-al. 1994] Anton, A.I.; McCracken, M. W.; Potts, C.: "Goal Decomposition and Scenario Analysis in Business Process Reengeeniering". In *Proceedings of the 6th CAiSE Conference*, CAISE 1994. Pages: 94-104.

[Ayala-Botella-Franch 2005] Ayala, C.P; Botella, P.; Franch, X.: "On Goal-Oriented COTS Taxonomies Construction". In *Proceedings 4th International Conference on COTS-Based Software Systems*, ICCBSS 2005. Springer-Verlag, LNCS 3412. Pages: 90-100.

[Babar-Gorton 2004] Babar, M.A.; Gorton, I.: "Comparison of scenario-based software architecture evaluation methods". In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, APSEC 2004. Pages: 600-607.

[Balk-Kedia 2000] Balk, L.D.; & Kedia, A.: "PPT: A COTS integration case study". In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE 2000. Pages: 42-49.

[Bao-Horowitz 1996] Bao, Y.; Horowitz, E.: "Integrating through user interface: A flexible integration framework for thirdparty software". In Proceedings of COMPSAC 1996. Pages: 336-342.

[Bass 1998] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.

[Bastos-Castro 2004] Bastos, L.R.D.; Castro, J.F.B.: "Enhancing Requirements to derive Multi-Agent Architectures". In *Anais do Workshop em Engenharia de Requisitos*, WER 2004.
[Bastos-Castro 2003] Bastos, L.R.D.; Castro, J.F.B.: "Integration between Organizational Requirements and Architecture" In *Anais do Workshop em Engenharia de Requisitos*, WER 2003. Pages: 124-139.

[Bleinstein-et-al. 2005] Bleistein, S.J.; Cox, K.; Verner, J.: "Strategic alignment in requirements analysis for organizational IT: an integrated approach". In *Proceedings of the 2005 ACM symposium on Applied computing*, SAC 2005. Pages: 1300-1307.

[Boehm-et-al. 1998] Boehm, B.; Abts, C.; Bailey, E.: "COCOTS Software Integration Cost Model: an Overview". In Proceedings of the California Software Symposium, 1998.

[Boehm-Abts 1999] Boehm, B.; Abts, C.: "COTS Integration: Plug and Pray?" IEEE Computer, Vol. 32, No. 1; January 1999. Pages: 135-138.

[Bresciani-et-al. 2004] Bresciani, P.; Perini, A.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J.: "Tropos: An Agent-Oriented Software Development Methodology". In Journal of Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers, Vol. 8, Issue 3, 2004. Pages: 203-236.

[Briand 1998] Briand, L.C.: "COTS Evaluation and Selection". In *Proceedings International Conference on Software Maintenance*, 1998. Pages: 222-223

[Brownsword-et-al. 2000] Brownsword, L.; Oberndorf, T.; Sledge C.A. "Developing New Processes for COTS-Based Systems". IEEE Software , Vol. 17, No. 4; July-August 2000. Pages: 48-55.

[deBruin-vanVliet 2002] de Bruin, H. ; van Vliet, H. : "Top-Down Composition of Software Architectures". In *Proceeding of the 9th IEEE International Conference on Engineering of Computer-Based Systems*, ICCBSS 2002. Springer Verlag, LNCS 2255. Pages: 147-158.

[Brynjolfsson-et-al. 1997] Brynjolfsson, E.; Renshaw, A.A.; van Alstyne, M.: "The Matrix of Change: A Tool for Business Process Reengineering". Sloan Management Review, Winter 1997. btrconsulting.net

[Burmeister 1996] Burmeister, B.: "Models and Methodology for Agent-Oriented Analysis and design". *Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems,* 1996. DFKI Document D-96-06.

[Carney-Long 2000] Carney D.; Long F.: "What Do You Mean by COTS? Finally a Useful Answer". IEEE Software, Vol. 17, No. 2, March/April 2000.

[Cernuzzi-Rossi 2002] Cernuzzi, L.; Rossi. G.: **"**On the evaluation of agent oriented modeling methods". In *Proceedings of Agent Oriented Methodology Workshop*, 2002.

[Chung-Nixon-Yu 1999] Chung, L.; Gross, D.; Yu, E.: "Architectural Design to Meet Stakeholder Requirements". In Proceedings of the First Working IFIP Conference on Software Architecture, WICSA 1999. Pages: 545-564.

[Chung-Nixon-Yu 2000] Chung, L.; Nixon, B.; Yu, E.; Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[Chung-Subramanian 2001] Chung, L.; Subramanian, N.: "Process-Oriented Metrics for Software Architecture Adaptability". In *Proceedings of 9th IEEE International Conference on Requirements Engineering*, RE 2001. Pages: 310-311.

[Chung-Cooper 2004] Chung, L.; Cooper, K. "Defining Goals in a COTS-Aware Requirements Engineering Approach". System Engineering, Vol. 7, No.1, 2004.

[Clements 1996] Clements, P.C.: "A Survey of Architecture Description Languages". In *Procedings of theEighth International Workshop on Software Specification and Design*,1996.

[Clements-et-al. 2002] Clements, P. ; Kazman, R.; Klein, M. : *Evaluating Software Architectures. Methods and Case Studies*. ISBN 0-01-70482-X. Addison-Wesley, 2002.

[Collinot-et-al. 1996] A. Collinot, A. Drogoul, P. Benhamou. "Agent oriented design of a soccer robot team". In *Proceedings of the Second International Conference on Multi-Agent Systems*, ICMAS-96). Pages: 41-47.

[Coyette 2003] Coyette, A.: *The SkwyRL-Agent Architectural Framework: Developping An E-Business Application*.
http://www.isys.ucl.ac.be/skwyrl/emedia/Files/report.pdf

[Curtis-et-al. 1992] B. Curtis, B.; Kellner, M.I.; Over, J.: "Process Modelling". Communications of the ACM, Vol. 35, No. 9, September 1992.

[Dam-Wnikoff 2003] Dam, K.H.; Wnikoff, M.; "Comparing Agent-Oriented Methodologies". In *Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems*, AAMAS 2003.

[Dardenne-et-al. 1993] Dardenne, A.; vam Lamswerde, A.; Fickas, S.: "Goal-Directed Requirements Acquisition". Science of Computer Programming, Vol. 20, 1993. Pages: 3-50.

[Davenport-Stoddard 1994] Davenport, T.H.; Stoddard, D.B.: "Reengineering: business change of mythic proportions?". MIS Quaterly, Vol. 18, Issue 2, June 1994. Pages: 121 - 127.

[Dobrica-Niemelä 2002] Dobrica, L.; Niemelä, E.; "A survey on software architecture analysis methods". IEEE Transactions on Software Engineering, Vol. 28 , Issue 7, July 2002. Pages: 638 – 653.

[Eertink-et-al. 1999] Eertink, H.; Janssen, W.; Luttighuis P.O.; Teeuw, W.; Vissers, C.: "A Business Process Design Language". In *Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems -Volume I*, 1999. Springer Verlag, LNCS 1708. Pages: 76-95.

[Estrada-et-al. 2003] Estrada, H., A. Martínez, A., Pastor, O.: "Goal-based business modeling oriented towards late requirements generation". In Proceedings of the 22nd International Conference on Conceptual Modeling, 2003. Springer-Verlag, LNCS 2813. pp. 277-290.

[Franch-Carvallo 2003] Franch, X.; Pablo Carvallo, J.; "Using quality models in software package selection" IEEE Software, Volume 20, Issue 1, Jan-Feb 2003. Page(s):34-41.

[Franch-Maiden 2003] Franch, X.; Maiden, N. "Modelling Component Dependencies to Inform their Selection". In *Proceedings 2nd International Conference on COTS-Based Software Systems*, ICCBSS 2003.

[Franch 2005] Franch, X.: "On the Lightweight Use of Goal-Oriented Models for Software Package Selection". In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering*, CAISE 2005. Springer-Verlag, LNCS 3520. Pages: 551-566.

[Fuentes-et-al. 2004] Fuentes, R.; Gómez, J.J.; Pavón, J.: "Social Analysis of Multi-agent Systems with Activity Theory". In Current Topics in Artificial Intelligence: 10th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2003, and 5th Conference on Technology Transfer, TTIA 2003. Springer-Verlag, LNCS 3040.

[Garlan-et-al. 1995] Garlan, D.; Allen, R.; Ockerbloom, J.: "Architectural Mismatch: Why Reuse Is So Hard". IEEE Software, Vol. 12, No. 6, 1995. Pages: 17-26.

[Garlan-et-al. 1995] Garlan, D.; Allen, R.; Ockerbloom, J.: "Architectural Mismatch or Why it's hard to build systems out of existing parts". In *Proceedings of International Conference on Software Engineering*, ICSE 1995. Pages: 179-185.

[Garlan-et-al. 97] Garlan, D.; Monroe, R.; Wile, D.: "Acme: An Architecture Description Interchange Language". In *Proceedings of CASCON 97*. Pages: 169-183.

[Giaglis 2001] Giaglis, G.M.: "A taxonomy of business process modelling and information systems modelling techniques". *International Journal of Flexible Manufacturing Systems*, Vol. 13, No. 2, 2001. Page(s):209-228.

[Goguen-Linde 1993] Goguen, J.A.; Linde, C. "Techniques for Requirements Elicitation" Proceedings IEEE Symposium on Requirements Engineering, San Diego, California (IEEE Computer Society Press 1993) Pages: 152-164.

[Goetz-Rupp 2003] Goetz, R., Rupp, C.: "Psychotherapy for System Requirements". In *Proceedings of the 2nd IEEE International Conference on Cognitive Informatics*, 2003. pp. 75-80.

[Gorton-Liu 2002] Gorton, I. & Liu, A.: "Streamlining the acquisition process for large-scale COTS middleware components". In *Proceedings of the 1st International Conference on COTS-Based Software Systems*, ICCBSS 2002, LNCS 2255. Pages: 122-131.

[Grant 2002] Grant, D.: "A Wider View of Business Process Redesign". Communications of the ACM. February 2002/Vol. 45, No. 2. Page(s) 85-90.

[Green 1994] Green, S.: "Goal-Driven Approaches to Requirements Engineering" Technical Report DoC TR-93-42 1994, Imperial College of Science, Technology and Medicine, Department of Computing Technical Report, London, UK, 1994.

[GRL-web] GRL web page, http://www.cs.toronto.edu/km/GRL/, last accessed April 2005.

[Grover-et-al 1994] Grover, V.; Fiedler, K.D.; Teng, J.T.C.: "Exploring the success of information technology enabled business process reengineering". IEEE Transactions on Engineering Management, Vol. 41, No 3, August 1994.

[Grover-Malhotra 1997] V. Grover, M.K. Malhotra. "Business process reengineering: A tutorial on the concept, evalution, method, technology and aplication". Journal of Operations Management, Vol. 15, 1997. Elsevier. Pages: 193-213.

[Gunasekaran-Nath 1997] Gunasekaran A.; Nath B.: "The role of information technology in business process reengineering". International Journal of Production Economics, Vol. 50, No. 2, June 1997. Elsevier Science. Pages: 91-104.

[Guo 2000] Guo, J.: "Interoperability technology assessment". Elsevier Science, Electronic Notes in Theoretical Computer Science, Vol. 65, No. 4, 2000.
[Hammer-Champy 93] Hammer, M.; Champy, J.A. *Reengineering the Corporation: a Manifesto for Business Revolution*. Harper Business, New York, 1993.

[Iglesias-et-al. 1998] C.A. Iglesias, M. Garijo, J.C. González, J.R. Velasco. "Analysis and design of multiagent systems using MAS-CommonKADS". INTELLIGENT AGENTS IV: Agent Theories, Architectures, and Languages, Springer Verlag, 1998.

[Iglesias-et-al 1999] C.A. Iglesias , M. Garijo, J.C. González. "A Survey of Agent-Oriented Methodologies". In *Proceedings of the 5th International Workshop on Intelligent Agents V. Agent Theories, Architectures, and Languages*, ATAL 1998. Springer-Verlag. Pages: 317 – 330.

[ISO/IEC 9126-1] ISO/IEC Standard 9126-1 Software Engineering – Part 1: Quality Model, 2001.

[*i**-web] *i** web page, http://www.cs.toronto.edu/km/istar/, last accessed January 2006.

[*i**-wiki] *i** web page, http://istar.rwth-aachen.de/, last accessed January 2006

[Janssen-et-al. 1997] Janssen, W.; Jonkers, H.; Verhoosel, J.P.C.: "What makes business processes special? An evaluation framework for modelling languages and tools in Business Process Redesign". In *Proceedings 2nd CAiSE/IFIP 8.1 international workshop onevaluation of modelling methods in systems analysis and design*, Barcelona, June 1997.

[Jarzabek-Tok 1996] Jarzabek S.; Tok W.L.: "Model-based support for business re-engineering". Information and Software Technology, Volume 38, Number 5, May 1996. Elsevier Science. Pages: 355-374.

[Jennings-Campos 1997] Jennings, N.R., Campos, J.R.: "Towards a social level characterisation of socially responsible agents." In IEEE proceedings on software engineering, 1997. Page(s) 144:11—25.

[Jennings 1998] Jennings, N.R.: "A Roadmap of Agent Research and Development". Autonomous Agents and Multi-Agent Systems. Volume 1 , Issue 1, 1998. Pages: 7 – 38.

[Jennings-Wooldridge 99] Jennings, N.R.; Wooldridge, M: "Agent-Oriented Software Engineering".

[Jennings 2000] Jennings, N.R.: "On Agent-Based Software Engineering". Artificial Intelligence, Vol. 117, No. 2, 2000. Pages: 277–296.

[Jones-Maiden 2004] Jones, S., Maiden, N.A.M.: "RESCUE: An Integrated Method for Specifying Requirements for Complex Socio-Technical Systems". Book chapter in *Requirements Engineering for Sociotechnical Systems*, Idea Group Inc., 2004.

[Kaiya-et-al. 2002] H. Kaiya. H. Horai. M. Saeki. "AGORA: Attributed Goal-Oriented Requirements Analysis Method". In Proceedings of the IEEE Joint International Conference on Requirements Engineering, RE 2002. Pages: 13-22

[Katzenstein-Lerch 2000] G. Katzenstein, F.J. Lerch: "Beneath the surface of organizational processes: a social representation framework for business process redesign". ACM Transactions on Information Systems, Vol. 18, No. 4, 2000. Pages: 383-422.

[Kavakli-Loucopolos 2004] Kavakli, E.; Loucopoulos, P.: "Goal Modeling in Requirements Engineering: Analysis and Critique of Current Methods". Information Modeling Methods and Methodologies. Idea Group, 2005.

[Kazman-et-al. 94] R. Kazman, L.J. Bass, M. Webb, G.D. Abowd: "SAAM: A Method for Analyzing the Properties of Software Architectures". *Proceedings of the 16th International Conference on Software Engineering*, ICSE 1994. Pages: 81-90.

[Kazman 1996] Kazman, R.; Abowd, G.; Bass, L.; Clements, P.: "Scenario-based analysis of software architecture". IEEE Software 13(6), 1996. Pages: 47-55.

[Kettinger-et-al. 1993] Kettinger, W.J.; Guha, S.; and Teng, J.T.C.: "Business process reengineering: Building the foundation for a comprehensive methodology". Journal of Management Information Systems, Vol. 10, No. 1, Summer 1993. Pages: 13-22.

[Kettinger-et-al. 1997] Kettinger, W.J.; and Teng, J.T.C.; Guha, S.; "Business Process Change: A Study of Methodologies, Techniques, and Tools". MIS Quarterly, 1997.

[Kinny-et-al. 2006] David Kinny, Michael Georgeff, and Anand Rao. "A methodology and modelling technique for systems of BDI agents." In W. van der Velde and J. Perram, editors, Agents Breaking. In *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, MAAMAW 1996. Springer-Verlag, LNCS 1038.

[Klein 1994] M.M. Klein: "Reengineering methodologies and tools". Information Systems Management, 11(2), 1994.

[Klein-Kazman 1999] Klein, M. and R. Kazman, "Attribute-Based Architectural Styles," Technical Report CMU/SEI-99-TR-022, Soft Engineering Institute, Carnegie Mellon University, 1999

[Kolp-et-al. 2001] Manuel Kolp, Paolo Giorgini, John Mylopoulos: "A Goal-Based Organizational Perspective on Multi-agent Architectures". ATAL 2001. Pages: 128-140.

[Kolp-et-al. 2003] Manuel Kolp, Paolo Giorgini, John Mylopoulos: "Organizational Patterns for Early Requirements Analysis". CAiSE 2003. Pages: 617-632.

[Kontio 1996] Kontio, J.: "A Case Study in Applying a Systematic Method for COTS Selection". In *Proceedings 18th International Conference on Software Engineering*, IEEE Computer Society Press.

[Koubarakis-Plexousakis 2000] Manolis Koubarakis, Dimitris Plexousakis. "A Formal Model for Business Process Modeling and Design". In *Proceedings of the 22nd Conference on Advanced Information Systems Engineering*, CAISE 2000. Pages: 142-156.

[Kueng-Kawalek 1997] Kueng P.; Kawalek P.: "Goal-based business process models: creation and evaluation". Business Process Management Journal, Volume 3, Number 1, January 1997. Emerald Group Publishing Limited. Pages: 17-38.

[Kunda-Brooks 1999] Kunda, D.; Brooks, L. "Applying Socio-Technical Approach for COTS Selection". In Proceedings UK Academic Information Systems Conference, 7-9 April 1999, University of York. Pages: 552-565

[Kozaczynski 2002] Kozaczynski, W.: "Requirements, Architectures and Risks". In *Proceedings of the 10th IEEE International Requirements Engineering Conference*, RE 2002. Pages: 6-7.

[vanLamsweerde-et-al. 1992] van Lamsweerde, A.; Darimont, R.; Massonet, P.: "The Meeting Scheduler Sytem – Problem Statement". 1992.
http://www.lore.ua.ac.be/Teaching/SSPEC2LIC/MeetingScheduler.pdf.

[vanLamsweerde 2001] van Lamsweerde, A. "Goal-Oriented Requirements Engineering: A Guided Tour" Proceedings 5th IEEE International Symposium on Requirements Engineering, Toronto, August 2001. Pages: 249-263.

[Lauesen 2004] Lauesen, S.: "COTS Tenders and Integration Requirements". In *Proceedings of the 12th IEEE International Requirements Engineering Conference* (RE'04). Pages: 166-175.

[Lee 1993] Lee, J.: "Goal Based Process Analysis: A Method for Systematic Process Redesign". In *Proceedings of the Conference on Organizational Computing Systems*, COOCS 1993. Pages: 196-201.

[Maiden-Ncube 1998] Maiden, N.; Ncube, C.; "Acquiring Requirements for COTS Selection". IEEE Software Vol. 15, No. 2, March/April 1998.

[Medvidovic-Taylor 1997] N. Medvidovic, R.N. Taylor. "A Framework for Classifying and Comparing Architecture Description Languages". *In Proceedings of the Sixth European Software Engineering Conference*, 1997. Springer-Verlag, LNCS 1301. Pages: 60-76.

[Medvidovic-Taylor 2000] Medvidovic, N.; Taylor, R.N.; "A classification and comparison framework for software architecture description languages". IEEE Transactions on Software Engineering. Volume 26, Issue 1, January 2000. Pages: 70-93.

[Meyers-Oberndorf 2002] Meyers, C.B.; Oberndorf, P. "Managing Software Acquisition". SEI Series in Software Engineering, 2002

[Morisio-Tsoukias 1997] Morisio, M.; and Tsoukias, A. "IusWare: A Methodology for the Evaluation and Selection of Software Products", IEEE Software Engineering Vol. 144, No. 3, 1997, Pages: 162-174

[Morisio-et-al. 2000] Morisio, M.; Seaman, C.B.; Parra, A.; Basili, V.; Kraft, S.; and Condon, S. "Investigating and Improving a COTS-Based Software Development Process". *Proceedings of the 22nd International Conference on Software Engineering*, ICSE 2000. Pages: 32-41.

[Ncube-Maiden 1999] Ncube, C.; Maiden, N. "Guiding Parallel Requirements Acquisition and COTS Software Selection". In *Proceedings of the Fourth IEEE International Symposium on Requirements Engineering*, RE 1999, Pages: 133-140

[Neto-et-al. 2004] Neto, G.C., Gomes, A.S., Castro, J.B.: "Mapeando Diagramas da Teoria da Atividade em Modelos Organizacionais Baseados em *i**". In *Proceedings of the 7th Workshop em Engenharia de Requisitos*, 2004, pp 39-50.

[Newell 1982] A. Newell. *The knowledge level*. Artificial Intelligence, Vol. 18, 1982. Pages: 87-127.

[Oberndorf-Brownsword 1997] Oberndorf, P.; Brownsword, L. "Are You Ready for COTS?" Software Institute Engineering. August 1997.

[O'Malley-DeLoach 2001] S. A. O'Malley and S. A. DeLoach. "Determining when to use an agent-oriented software engineering". In *Proceedings of the Second International Workshop On Agent-Oriented Software Engineering*, AOSE-2001. Springer-Verlag 2222/2002. Page(s) 188-205.

[Pavan-et-al. 2003] P. Pavan, N.A.M. Maiden, X. Zhu. "Towards a Systems Engineering Pattern Language: Applying *i** to Model Requirements-Architecture Patterns". In *Proceedings of STRAW* 2003.

[Potts-et-al. 1994] Potts, C.; Takanashi, K.; Antón, A. "Inquiry-Based Requirements Analysis". IEEE Software, Vol. 11, No. 2, March 1994.

[Rolland-et-al. 2004] C. Rolland, C. Salinesi, A. Etien: "Eliciting gaps in requirements change". Requirements Engineering. Vol. 9, No. 1, February 2004. Pages: 1-15.

[Rugg-et-al. 1992] Rugg, G, Corbrigge, C. Major, N.P.; Burton, A.M.; Shadbolt, N.R. "A Comparison of Sorting Techniques in Knowledge Elicitation" Knowledge Acquisition, Vol. 4, No. 3, 1992. Pages: 279-291.

[Rugg-McGeorge 1997] Rugg, G, McGeorge, P. "The Sorting Techniques: a Tutorial Paper on Card Sort, Picture Sorts and Item Sorts", Expert Systems, Vol 14, No.2. May 1997.

[RUP] Jacobson, I.; Booch, G.; Rumbaugh, J. " The Unified Software Development Process". Addison Wesley Longman, Reading, MA, 1999.

[Sabas-et-al. 02] A. Sabas, S. Delisle, M. Badri. "A Comparative Analysis of Multiagent System Development Methodologies: Towards a Unified Approach", *Third International Symposium "From Agent Theory to Agent Implementation" (AT2AI-3), Sixteenth European Meeting on Cybernetics and Systems Research*, Vienne (Autriche), 2-5 avril 2002.

[Santander-Castro 2002] Santander, V.F.A.; Castro, J.F.B.: "Deriving Use Cases from Organizational Modeling". In *Proceedings of the 10th IEEE Requirements Engineering Conference*, 2002. Pages: 32-39.

[Saur-et-al. 2000] Saur, L.D.; Clay, R.L. & Armstrong, R.: "Meta-component architecture for software interoperability". IEEE Computer, November, 2000, Pages: 75-84.

[Sai-Franch-Maiden 2004] V. Sai, X. Franch, N.A.M. Maiden: "Driving Component Selection through Actor-Oriented Models and Use Cases". *Proceedings of the Third International Conference on COTS-Based Software Systems*, ICCBSS 2004. Springer Verlag, LNCS 2959. Pages: 63-73.

[Schreiber-et-al. 1994] A. Th. Schreiber, B. J. Wielinga, J. M. Akkermans, andW. Van de Velde. "CommonKADS: A comprehensive methodology for KBS development. Deliverable DM1.2a KADSII/M1/RR/UvA/70/1.1, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels, 1994."

[SEI] Software Engineering Institute. COTS-Based System Initiative (CBS) http://www.sei.cmu.edu/cbs/

[SEI-a] Software Engineering Institute. COTS-Based System Initiative (CBS) http://www.sei.cmu.edu/cbs/overview2.html

[SEI-b] Software Engineering Institute. COTS-Based System Initiative (CBS) http://www.sei.cmu.edu/cbs/overview.html

[Shaw 1991] Shaw M.; 1991, 'Heterogeneous Design Idioms for Software Architecture', In *Proceedings of the Sixth International Workshop on Software Specification and Design*. IEEE Computer Society Press, Pages: 158-165.

[Söderström-et-al. 2002] E. Söderström, B. Andersson, P. Johannesson, E. Perjons, B. Wangler1. "Towards A Framework for Comparing Process Modelling Languages" In Proceedings of the 14th International Conference on Advanced Information Systems Engineering, CAISE 2002. Springer Verlag, LNCS 2348.
Pages: 600 – 611.

[Sturm-Shehory 2003] A. Sturm, O. Shehory. "A Framework for Evaluating Agent-Oriented Methodologies". In *Proceedings of the Agent-Oriented Information Systems, 5th International Bi-Conference Workshop*, AOIS 2003. Pages: 94-109.

[Sudeikat-et-al. 2004] J. Sudeikat, L. Braubach, A. Pokahr, W. Lamersdorf. "Evaluation of Agent - Oriented Software Methodologies - Examination of the Gap Between Modeling and Platform". In *Proceedings of the Fifth International Agent-Oriented Software Engineering Workshop*, AOSE 2004. Pages: 126—141

[Teng-et-al. 98] J.T.C. Teng, S.R. Jeong, V. Grover. "Profiling successful reengineering projects". Communications of the ACM. Volume 41, Issue 6, June 1998. Pages: 96 – 102.

[Tveit 2001] Tveit, A.: "A Survey of Agent-Oriented Software Engineering". In *Proceedings of the First NTNU CSGS Conference*, 2001.

[Tropos-web] TROPOS web page, http://www.troposproject.org/, last accessed April 2005.

[UML] UML 2.0 Specifications http://www.uml.org/, last accesed February 2005

[Vestal 1993] S. Vestal. A Cursory Overview and Comparison of Four Architecture Description Languages. Technical Report, Honeywell Technology Center, February 1993.

[Voas 1998] Voas, J. "The Challenges of Using COTS Software in Component-Based Development". IEEE Computer, Vol. 31 No.6, June 1998. Pages: 44-45.

[Warboys-et-al. 2005] B. Warboys, B. Snowdon, R.M. Greenwood, W. Seet, I. Robertson, R. Morrison, D. Balasubramaniam, G. Kirby, K. Mickan. "An Active-Architecture Approach to COTS Integration". IEEE Software, Volume 22, Issue 4, July-Aug. 2005 Pages: 20-27.

[Wooldridge 1997] Wooldridge, M.: "Agent-based software engineering". IEEE Proceedings – Software, Vol. 144, No. 1, 1997 Pages: 26-37.

[Wooldridge-et-al. 2000] M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia methodology for agent-oriented analysis and design". Journal on Autonomous Agents Multi-Agents Systems.; vol. 3, no. 3, 2000.

[Yakimovich-et-al. 1999] Yakimovich, D.; Bieman, J.M. & Basili, V.R.: "Software architecture classification for estimating the cost of COTS integration". In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, ACM, Pages: 296-302.

[Yu-et-al. 1996] ESK Yu, J Mylopoulos, Y Lesperance. "AI Models for Business Process Reengineering". IEEE Expert: Intelligent Systems and Their Applications, 1996.

[Yu-Cysneiros 2002] Eric S. K. Yu, Luiz Marcio Cysneiros: "Agent-Oriented Methodologies - Towards a Challenge Exemplar". In *Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems*, AOIS 2002.

[Yu-Cysneiros 2002] E. Yu, L.M. Cysneiros, "Agent-Oriented Methodologies – Towards A Challenge Exemplar". *Third International Symposium "From Agent Theory to Agent Implementation" (AT2AI-3), Sixteenth European Meeting on Cybernetics and Systems Research*, Vienne (Autriche), 2-5 avril 2002.

[Yu-Mylopoulos 1994] Yu, E.; Mylopoulos, J. "Using Goals, Rules, and Methods to Support Reasoning in Business Process Reengineering". In *Proceedings of the 27th Hawaii International Conference on System Sciences*, *Vol. IV: Information Systems: Collaboration Technology Organizational Systems and Technology*, January 1994.

[Yu 1995] Yu, E.: *Modelling Strategic Relationships for Process Reengineering*, PhD. thesis, University of Toronto, 1995.