1. Project rationale

The aim of this project is to provide a methodology and its associated tools in order to support the integration of disparate business applications that have not necessarily been designed to coexist. Inspiration comes from real concerns that are the result of an investigative effort on the part of some of the research partners in this consortium; the object of the investigation was the identification of mainstream ICT problems with a representative forum of Belgian enterprises (large and small) that rely on information technology for their critical business activities.

Part of what we propose to investigate is covered by the newly named discipline of Enterprise Application Integration; another part is covered by re(verse) engineering; however, our ambitions reach further.

At the roots of the ARRIBA project are two driving forces. On the one hand we have a consortium of research groups that have been active in the field of software engineering and more particularly in re(verse) engineering, software evolution and software architectures. These groups have a fairly long-standing history of cooperation and involve 3 Flemish and 2 non-Flemish sites (the latter two play a supporting role): their composition, expertise and experience (both academic and industrial) is described in this section. These research groups feel confident that they can join forces and tackle the new and ambitious problem domain targeted in the ARRIBA proposal.

On the other hand, we have the already mentioned and recently created forum of Belgian enterprises interested in a joint initiative to identify generic problems and likewise generic solutions plaguing their ICT component. This forum has the form of a foundation hosted by what could best be described as a collective spin-off of the five Flemish computer science departments.

These two contributing agents – the first providing the content of the ARRIBA project, the second providing the context – will guarantee the correct identification of the problem setting, the marshalling of the required scientific-technical resources and the proper channeling of the results to the business world.

The dynamics of modern business applications is characterized by a need for restructuring and integration at a much larger scale than was previously the case. This follows the restructuring and integration of the organizations themselves, as they strive to merge their activities and, hence, also their ICT infrastructure. In many cases, the major hurdle that needs to be taken consists of getting the software applications themselves to collaborate: in all but a few cases the documentation and support of these applications is insufficient (or even absent). Generally, the data repositories and the running programs themselves are the only true description of the information structures and applications they implement; hence, the only dependable documentation is the actual data (and the way in which it is handled) and the source code itself (with the exception of the truly pathological case where executable code and source code are out of sync — a situation which we will not consider here). We do not ignore the possible existence of design and support documentation; we simply state that (at best) its validity will need to be checked against the data and applications themselves.

Merging business applications will always require the application of human insight and experience; but we can provide means (methods and tools) to address many of the repetitive, low level but large scale issues. The level at which this human-machine integration framework should operate is the architectural level. In general, the architecture of a (business) application is concerned with the part of the implementation that describes the more abstract structure of the application and the data that it acts upon, together with essential structural issues that cannot be directly expressed in the various schemata and program code. Typically, this last part is a reflection of pertinent domain knowledge and/or design considerations. The architecture of an application has proved to be an interesting level at which to reason about its design, evolution and even comprehension; this would indicate that the architecture of, say, two independently developed (business) applications, is the level at which to start considering their coexistence.

It is our firm conviction that, although integration of business applications is identified as an important issue, the current EAI-inspired methods and tools fail to provide a satisfactory setting in which to proceed. They cover relatively simple concerns but do not effectively address the issues mentioned before; matters of scale, weak documentation, lack of consistency etc. are not handled in an appropriate way.

The user committee that is proposed in the ARRIBA project offers access to a relevant and representative spread of business cases where issues in integration of ICT resources – and possibly restructuring them in view of future co-evolution – are present. On the other hand, the academic partners can bring into play experience (generally instantiated in methodologies and tools) in the following domains: construction and extraction of software architectures, reverse engineering and reengineering of legacy systems, the co-evolution of the design and the implementation of software systems. In the proposed research project these are at least as relevant as the field described as *Enterprise Application Integration*. Although the tangible results (in terms of methods, tools and other artifacts) of the existing expertise in the consortium only addresses particular features of the ARRIBA objectives, they can be marshaled as basic building blocks for the proposed approach.

We should at all times keep in mind that a business application is the instantiation of a specific business process and that the merge of two applications is driven by the merge of the associated processes (and not the other way around). The ARRIBA project, although clearly technological in nature, is very much driven by business concerns. The user committee unites a number of organizations that are very much involved in the issues that are addressed by ARRIBA; they will function at least as a permanent reality check.

Programming Technology Lab [PROG] – Vrije Universiteit Brussel[1]

The Programming Technology Lab was founded in the mid eighties as a part of the Department of Mathematics. In the late eighties it became a major player in an actual Department of Computer Science. Currently, PROG hosts about 15 researchers who are financed by the university and mainly by projects and grants provided by the Flemish and Brussels government; these regularly occur in cooperation with industrial partners. PROG will assume the role of coordinator for the ARRIBA project.

The research carried out at PROG can be subdivided into three (non-disjoint) fields:

Software (co-)evolution: first, there is a group of researchers whose long term goal is the development of languages, abstractions and tools that allow the easy generation of code based on (architectural and design) knowledge about that code. To this extent, a technique called Declarative Meta Programming (DMP) was developed. The general idea behind DMP is to have a declarative language (like Prolog) that reasons about and acts upon (i.e. generates and changes) object-oriented code (e.g. Smalltalk). This allows one to write a logic program that represents (the structure of) an object-oriented program. Running the logic program (i.e. querying) will thus result in a reasoning process that reasons about existing code residing in a repository and injects new code into that repository. In order to accomplish this, a system called SOUL has been developed. SOUL is an implementation of Prolog that reasons about and generates Smalltalk code. Although SOUL is used by the different members of this research group as a common technological platform, every researcher tries to develop his/her different applications of DMP. Examples of such applications currently under inspection are aspect-oriented programming, generative programming, compliance checking between architectural and design knowledge on the one hand and code on the other hand, separating control and user-interface code (and generating the necessary glue). Representative publications are: [DDMW00, MMWW00].

Knowledge based software: a second group of people can be categorized as trying to recover existing techniques from the field of artificial intelligence and apply them onto the development cycle of object-oriented software. A first example of this kind of research is the usage of ontologies to set up a common vocabulary to be used by all the people participating in the construction and maintenance of a system. Representative publications are: [WDVP00, DW00]. A second example is to use expert system technology to see how far we can get in establishing an automated link between the documents (requirement specifications, code, design graphs,...) resulting from different phases of the life cycle. This link is necessary because evolution (at all levels) often changes one of those documents and we want the other documents to stay synchronized. The final example is to use knowledge driven techniques to do so-called code mining. Knowledge about code and design is used to steer the extraction of abstract descriptions (design patterns, architectures) out of flat code. Representative publications are: [DDW99], [TDM99], [WT01].

[1] http://prog.vub.ac.be/

Language engineering: the final group of researchers is interested in the design of object-oriented programming languages and their implementations (and is less relevant to this proposal). In this group, new object-oriented programming language abstractions and efficient techniques to implement them are the topic of investigation. One of the current research efforts is the development of a prototype-based language to ease the teaching of object-oriented software construction. Another example is the construction of a (prototype based) agent system that allows one to program autonomous agents that roam a wide-area network. Yet another example is the quest for language abstractions that help structuring mobile applications. Representative publications are: [VVD99], [VD00].

Information Technology [INTEC] – Universiteit Gent[2]

The University of Ghent, through the Department of Information Technology (INTEC) and its Software Engineering group, contributes concrete experience in the analysis and design of large administrative systems. Research has been focused in recent years to introduce sound ICT technology in existing or emerging enterprises. A bit in contrast to the other partners, but highly complementary to their efforts, this group has invested heavily in engineering projects that are state of the art, and have been made operational by its industrial partners. Those projects were focused on introducing and using the latest technology in realistic business environments. Object orientation and the integration of legacy software and legacy data has been the major concern, together with a bootstrapping environment for the partners concerned in order to reach a better technology level.

To name a few, and also to highlight the industrial relevance of the experience:

- The Belgian Senate[3]: in this project INTEC has designed the architecture of the total ICT infrastructure of the Belgian Senate, involving the latest technology in OO, data management and web publishing
- A sound architecture for the federal network intranet Fedenet and its public interface[4] and for the highly confidential decision support system for the Belgian government.
- The student administration of the University of Ghent[5], now in use in a number of national and international universities.
- Training of the Belgian Army in OO, with implementation of a (NATO secret) new messaging system Hermes on board of the ships of the Belgian Navy.
- Analysis, design and implementation of the critical parts of one of the most successful E-business systems in Europe, dealing with real-time auctions of fresh sea products over the Internet[6]. This application received the Trends award for excellence in 2001.
- Structural and architectural work around a number of highly publicized and innovative projects from INNO.COM[7]: Toyota Europe, Banksys and its mission-critical heart beat system, E-business efforts and so on.
- Probing the introduction of data management, role model based architecture and J2EE architecture through Aspect Oriented Programming in the highly competitive world of Toyota Europe.

Fundamental research has been coupled to those projects, and has learned from the projects in focusing on real life issues, mainly centred around data management, IT architecture and basic tools around it.

[2] http://www.intec.rug.ac.be/
[3] http://www.senate.be
[4] http://www.fgov.be
[5] http://www.rug.ac.be
[6] http://www.pefa.com
[7] http://www.inno.com

The results of this can be found in a number of publications. The global context of those publications is exposed in the highly successful book by Peter Heinckiens, in the prestigious Booch OO series of Addison-Wesley [H98].

Persistency of data through the coupling of object technology and relational database management systems is one of the key issues developed. Storage frameworks, and aspect oriented programming are the main focal point, coupled to declarative mechanisms for handling legacy data. It should be clear that the software engineering group is in the middle of the turmoil around Enterprise Application Integration EAI.

It also hosts the chairmanship of the Foundation "Bundelen van Breinen", where ICT and EAI issues are being discussed between ICT managers, offering a realistic view on the real life problems of ICT and EAI.

Lab On Reengineering [LORE] and Concurrency group – Universiteit Antwerpen [8]

The University of Antwerp contributes concrete experience with code mining techniques (reverse & reengineering patterns, code visualizers, metrics) and software modeling languages (UML, state-charts, petri-nets, actors, ...). Much of this experience stems from a tight collaboration with the SCG at UniBerne, and is available in a number of ways:

- a book addressing the problem of understanding and reengineering legacy software systems [D02];
- a number of tool prototypes that may deal with large scale legacy-code (*DUPLOC* [Duc99], *CODECRAWLER* [D99b], *MOOSE* [T01])
- an exchange format for integrating reverse- and reengineering tools (*FAMIX*) [D99a], [D01]
- studies on the applicability of metrics during reverse engineering ([D99c], [DDN00])
- a number of papers on the modelling of concurrent and distributed systems ([J96], [J99])
- participation in a number of European research networks concerning software modelling (COMPUGRAPH, APPLIGRAPH, GETGRATS, SEGRAVIS)

This experience is contributed by two research groups. On the one hand, the Lab on Reengineering (directed by Serge Demeyer), which is a brand new research group with a special interest in Object-oriented Reengineering. The group is directed by Serge Demeyer, who previously worked for the Software Composition Group at the University of Berne, hence the tight bounds between the two groups. LORE perceives reengineering as an essential phase in any software life-cycle because software systems —especially object-oriented ones— must adapt to changing requirements in order to remain successful. In particular, LORE investigates (a) software models, to express the present and future status of evolving software systems; (b) software metrics, to control the software evolution process; (c) reengineering and reverse engineering techniques, to transform and understand an existing system; and finally (d) tools, to help software engineers facing evolving systems and to empirically validate experimental findings.

On the other hand, the research group CONCURRENCY (directed by Dirk Janssens) has as its central research theme the study of concurrent software systems, more specifically the way to model their behaviour in a formal way. The group participated in a number of EU-projects and networks concerning graph-rewriting, international collaborations which are considered necessary for the success of the project. Originally, the research in this group was focusing on generation mechanisms for sets of graphs, but later on this focus shifted towards the development of models for parallel object-oriented systems and -more abstractly- true concurrency semantics and processes for rewriting systems. Recently, the insight grew that graph-rewriting systems can make an important contribution in the area of visual modeling techniques (e.g., fragments of UML) and software refactoring, two topics which are especially relevant in the context of this project.

[8] http://win-www.uia.ac.be/u/lore/

Département d'Ingénierie Informatique [INGI] – Université Catholique de Louvain[9]

People working at the *Département d'Ingénierie Informatique* (UCL) put their efforts on domains related to the design and implementation of reliable and efficient computing systems (with hard software component) through methods applicable to the industry.

The different aspects of the software engineering and software technology field are obviously handled on both theoretical and practical levels. Both software at application and systems levels are considered. Methods and techniques related to artificial intelligence and logic programming are playing an important role in the field of software technology. The Department of Computing Science and Engineering has developed a pole of expertise in this field for many years.

In particular, our research program on software engineering and software technology includes different activities that aim to contribute to the implementation of a professional and reliable software design and also large-scale and complex software packages. It consists of the following essential research activities:

Activity 1: Requirements engineering

Activity 2: Software development process engineering

Activity 3: Security critical software packages

Activity 4: Real-time systems design

Activity 5: Software evolution and co-evolution

We want to highlight the last activity here, as we think it is most related to the current project proposal and it shows an important overlap with the research performed at the other research institutes involved in this research project. This activity covers the following topics:

- *Co-evolution*: techniques and tools to keep software artefacts in different levels of the software life-cycle in sync when the software evolves
- *Declarative Meta Programming*: a technique/paradigm to build state-of-the-art (object-oriented) software development tools, and in particular tools that support co-evolution
- *Software evolution*: formal and semi-formal techniques to support software evolution for object-oriented software
- *Architectural conformance checking*: a specific case of co-evolution where we are interested in keeping the software architecture conform to the software implementation

Software Composition Group [SCG] – Universität Bern[10]

The Software Composition Group from the University of Bern has experience in components and component languages, and in the reengineering of legacy object-oriented systems. The contributions in this last area are especially relevant to the research described in this proposal. They consist of:

- The definition of a language independent meta model (*FAMIX*) [D99a,D01],
- the implementation of the reengineering environment *MOOSE* [T01],
- the evaluation of the usage of metrics in reengineering [D99c,DDN00],
- the definition of a novel approach for reverse engineering large applications [D99b,Duc01],
- the definition of new approaches for understanding classes (the class blueprints) [Lanz01],
- the language independent detection of duplicated code (*DUPLOC*) [Duc99],
- the use of dynamic information for extracting behavioural views (*GAUDI*) [Rich98a, Rich99a, Duc99a],

[9] http://www.info.ucl.ac.be/people/cvmens.html
[10] http://www.iam.unibe.ch/~scg/

- the evaluation of language independent refactorings [Tich00b],
- and the identification of reengineering patterns [D02].

In addition, the SCG is currently complementing its current query-based reengineering environment *MOOSE* with a logic programming engine (the language *SOUL*). The goal is to allow experiments regarding program understanding and extraction of software architectures.

It should be apparent from the description of the five academic partners in the consortium (3 active, 2 passive) that they have a long-standing history of cooperation. This has been the result of an effective interaction at the doctoral and post-doctoral level between researchers of the various groups. That is why a significant amount of expertise in the re-engineering of object-oriented systems is shared between the UIA and UniBern, and also why the technique of declarative metaprogramming is used at the UCL, UniBern and the VUB.

The Foundation 'Bundeling van Breinen'

The user committee is largely drafted from a recently created forum that regroups individuals representing academia and information technology dependent enterprises (represented by their ICT management). It was structured as a foundation named *'Bundeling van Breinen'* (*Meeting of minds*). Its activities are best described by extracts from their white paper:

> *"It is the aim of the Foundation to offer a qualitative and secure environment in which knowledgeable persons and their deputies can exchange ideas in order to come up with solutions for the problems they face and will face in the future. The Foundation is intellectually independent and is not guided by any commercial objective.*
>
> ...
>
> *In the longer term, the Foundation 'Bundeling van Breinen' may provide the possibility to the community of a 'sabbatical environment' for their key assistants.*
>
> ...
>
> *The Foundation 'Bundeling van Breinen' will not act as a consultancy solution provider for the external world."*

The current (non exhaustive) composition of the foundation is listed below; as can be seen, it represents a very diverse mix of organizations, which makes it the more surprising that the problems and needs appear to be fairly universal.

Acerta	Leuven
Banksys	Brussels
Belgacom	Brussels
Belgian Army	Brussels
De Post	Brussels
Elly Lilly	Mont-Saint-Guibert
Emmaüs vzw	Mechelen
European Commission	Brussels
ING - BBL	Brussels
Inno.com	Beerzel
Interbrew	Leuven
KBC Verzekeringen	Leuven
Mosaic Partnership Ltd.	Reddich (UK)
Toyota Motor Europe	Brussels

Timing constraints have kept us from getting all of the foundation's participants involved as members in the ARRIBA user committee. Since we define the committee as an open-ended structure, we expect this to happen shortly.

We will conclude this section with the remark that the members of the consortium have a history of involvement in industrial applications of their research. We can even safely state that the introduction of object technology in Flemish industry was to a fair extent enabled by early initiatives of several of the partners. It is no accident that MediaGeniX – one of the members of the user committee – originated as a high-tech spin-off of the Programming Technology Lab in order to channel object technology between research and the software industry. Also: the activities of the user committee will be moderated by one of its members, Inno.com. This was started 4 years ago as a collective spin-off of the five Flemish computer science departments; its technological content is driven by an academic advisory board that is regularly involved in solving challenging issues within the context of industrial software architecture projects.

2. Situating the project

2.1 Current state of the art

Software Architectures In [M00a] a fairly comprehensive overview is given of the state of the art in Software Architectures. In particular the way in which an expressive architectural language is proposed in which to describe software architectures and their mapping to some software implementation is of interest to the ARRIBA proposal. The language supports the declaration of multiple architectures, called architectural views, on the same software system. Each of those views focuses on a different aspect of the structure of that system. In addition to this architectural language, an algorithm is developed which reasons about the descriptions in the architectural language to automatically check conformance of the implementation of some software system to its architectural views. We feel this approach is of particular relevance to the ARRIBA project, which was the major reason to invite the UCL-partner into the consortium.

Another aspect of software architectures which is relevant is described in [MN95], [MNS95] and [D98]. These initiatives address the need for extracting architectural descriptions from software artifacts using lightweight mechanisms such as pattern matching and tagging, and leading to classifications.

Enterprise application integration (EAI) tries to deal with the problem of providing a user environment around a single point of access to a possibly very large number of applications, most of them dating from a pre–EAI era.

Most of the integration effort is done using so called new technology, based on networking and an object-oriented methodology. Web services are an emerging possibility for connecting legacy applications to newer functionality and to introduce a more consistent software architecture.

However, in most situations it is very difficult even to implement a strategy for EAI due to the chaos of many of the currently used systems, consisting of a wide spectrum of languages, tools, database systems, database schemes etc… In most cases no clear rules have been set forward in managing the data, even the mission critical data. More often than not data are replicated several times, with slight variations in their organization and use, and without the notion of a single point of data creation and strictly defined rules concerning their later use. Even simple atomic data like e.g. country codes, do not have a common definition, nor a single repository, although they play a major role in the overall operation of mission-critical international distribution systems.

This situation is aggravated by inadequate documentation, undocumented side effects of transactions, and misconceptions of data ownership and use.

Object oriented techniques have had a major impact on the design philosophy of new applications. At least two aspects have changed considerably over the last years: a shift away from data centric applications to business process centered implementation, and a different role for the DBMS systems.

The first aspect, often mistakenly identified as the multi-tier architecture, has become the main point of attention in the specification of loosely coupled systems, consisting of cooperating components. A typical example here is found in the Enterprise Java Beans architecture, supported by the J2EE environment, or in the .NET initiative of Microsoft. However, bridging the gap between existing ad hoc applications and the cleaner components, e.g. using connector components or software gateways is not trivial due to the cited problems.

The other aspect, the new role of the DBMS in an OO environment is a more subtle issue. The role of a DBMS from an OO point of view is persistency of state of the business objects, and easy querying. Application logic is handled by clearly defined interfaces and business rules. The semantics of state preservation do not impose rules or implied procedures on the data itself.

In a data centered application, the data flow and their transformation process forms the backbone of the operations. Semantics of use of the data is found in the code fragments that handle particular atoms of information, but also in the DBMS itself, where e.g. stored procedures and trigger mechanisms implement (part of) the business transactions. The problem here areas are clear: very tightly coupled systems, lack of a uniform approach, ad hoc multiplication of data base operations, often very dangerous ones.

There is currently no magic solution to those problems: a total rewrite is in most cases totally unrealistic, while software refactoring techniques may provide a gradual, but slow and resource consuming alternative.

Both approaches rely heavily on a correct description of the current applications. The only consistent source of information is the code body of the total set of applications, including database schemes, stored procedures and triggers. Analysis of that code must be automated, possibly rule driven. Among others, cut and paste reuse should be identified, but also the operations on the data and their flow within and across the applications. A consistent repository should result from it.

The field of *Enterprise Application Integration* is essentially market driven and is nearly exclusively developed by software vendors and consultancy companies (including the big 5). The academic involvement is very much restricted although potential links with the re-engineering research community are emerging. Mainstream EAI is very much (web) technology oriented and EAI-related publications are likewise technological rather than academic[11].

Re(verse) Engineering an excellent overview of this field and its relationship to industrial (legacy) applications is provided in [T01] and [Duc01]. This illustrates that significant expertise was built up at the Software Composition Group at the University of Berne; together with the longstanding collaboration between UniBern, UIA and VUB this led us to include this research group as a passive partner in the consortium. The idea is to call upon them for re(verse) engineering related consulting and temporary hosting of consortium researchers.

Aspect Oriented Software Development grew out of *aspect oriented programming* [KLM+97]. It is relevant for the *instrumentation* approach to be used in the ARRIBA project. The consortium has significant experience in this domain, has produced several contributions (see [BDMDV00], [BR00a]) and software artifacts that are well-known in the AOSD community. In particular, the *declarative metaprogramming* approach has proved very effective as a tool in which to express weaving procedures [BR00b].

Data Integration as mentioned before, a significant cross-section of Belgian information technology dependent enterprises reports crucial data-ownership concerns when investigating integration of business applications. Surprisingly, very little research happens in this field, although it would seem obvious that the confrontation between the "new" technology and the database community would have led to this. This lack of attention would seem to be confirmed by one of the few publications (and by one of the database community champions) addressing the issue [SH01].

2.2 Current R&D-activities in the domain of the project on an international basis

We list here several recent and on-going projects in the domain of this proposal. We do not intend to be exhaustive, but want to give an idea of current efforts.

Software Factories is an IWT-sponsored basic research program involving the Programming Technology Lab, N.V. MediaGeniX (one of the ARRIBA user committee members) and EDS Belgium. It investigates an approach to software assembly lines addressing a broad range of business situations using a technique dubbed "Loosely Coupled Components". The general idea is to support integration between components using on the one hand low-tech mechanisms such as XML/XSL and SOAP, and support for (task) ontologies on the other. This project is currently finalized, to the satisfaction of all participants [SoFa02].

[11] A typical example is the eAI journal (see http://www.eaijournal.com/) which is largely dominated by IBM

Several of the resulting conclusions and even some of the artifacts will be applicable to the ARRIBA project.

Compliance Checking is a (finalized) basic research program sponsored by the Brussels' Region involving the Programming Technology Lab and N.V. Getronics. The objective of this project was to be able to check evolving applications against their architectural specification and, conversely, to compute the impact of an architectural change on the associated implementation [CoCh01]. Various results of this project are applicable to the ARRIBA goals.

Foundations of Software Evolution[12] the goal of this research network (categorized as a *scientific research community* sponsored by the Flemish Science Foundation) is to come to a consistent set of formally-founded techniques and associated tools to support software developers with the common problems they encounter when developing large and complex software systems. The interdisciplinary character of this research network is reflected in the uniform use of mathematical formalisms as a foundation for concrete tools to support software evolution. This cross fertilization between mathematics and computer science is intended to lead to more robust tools that are more generally applicable without sacrificing efficiency. Another important issue is that we will not restrict ourselves to a particular phase of the software life-cycle. Instead, we will focus on techniques that are generally applicable throughout the entire software development life-cycle. This research network focuses on tools for:

Forward engineering**.**

Proactive tools to ensure consistency between implementation, design, analysis and software architectures (by prohibiting certain software changes); Retroactive tools to detect differences between implementation, analysis, design and architecture (when changes have occurred in at least one of these phases).

Reverse engineering**.**

Techniques to extract the relevant abstractions from source code in order to improve understanding of the global structure of a software system.

Re-engineering.

Techniques to restructure software in order to improve reusability, extensibility and maintainability.

Team Engineering.

Techniques to support software evolution when multiple developers change software simultaneously.

This research community counts all of the academic partners in the ARRIBA proposal among its members.

2.3 Background knowledge of the research consortium in the domain of the project

Reuse contracts constitute a methodology for managing reuse and evolution in object-oriented software development. They address the incremental/iterative development of reusable software components and models, and how to build highly customized applications based on these components and models. Much of the inspiration for developing reuse contracts has been drawn from the practical experience in developing object-oriented frameworks. It is a practical method. It builds on existing object-oriented analysis and design methods (in casu UML). Yet, it distinguishes itself from all other methodologies by its formal foundation and support for evolution: the way (reusable) components are reused is formally described in a contract (hence the name "reuse contract"). This forms the basis of tools for managing change and for checking the consistency between models and code.

We try to be realistic about the role of reuse in software development. With current day techniques, the investment in building reusable components seems only justifiable for systems that can be sold in a relatively broad market and that have a relatively well-known and stable problem domain (such as is the

[12] see: http://prog.vub.ac.be/poolresearch/FFSE/network.html

case with most generic application frameworks). For most companies, however, the goal is not to build reusable software libraries, but to provide solutions to customers more efficiently and with more quality. Therefore, reuse contracts adopt a highly evolutionary approach to the development of reusable components.

It is our experience that the importance of change and evolution are heavily under-estimated. Change and evolution are omnipresent in software development: maintenance, changing requirements, technological changes, customization, iterative and incremental development, ... However, change and evolution also hold many risks: erosion of the software architecture and documentation, proliferation of versions, change propagation, upgrade problems, merge conflicts, etc... Reuse contracts try to address some of these problems without burdening the software developer too much.

As a first validation of our ideas we started with the reuse of abstract classes through inheritance. There, a reuse contract documents the way a class is being reused by means of inheritance. The different ways a class can be subclassed is encoded by formal reuse modifiers extension, concretization and refinement, and their inverses cancellation, abstraction and coarsening. Formal rules were defined that describe what the impact of base class exchanges are on existing applications. This problem is usually referred to as the fragile base class problem. The results from our approach in this particular case can be found in [SDLM96].

As a second case we extended this basic model to incorporate also collaborations between different classes in a system. This work is described in [L97].

As a third validation, we have incorporated reuse contracts in the Unified Modeling Language (UML). This enabled us to apply reuse contracts to the many different models available in the UML. Reuse contracts can be developed for all these different models without needing to change the underlying methodology.

Software classifications Initiated by the PhD research of Koen De Hondt [D98], we started to investigate the use of *intentional software classifications* as an intuitive and lightweight means of modeling crosscutting concerns in the software. There is an urgent need for this because modularization mechanisms in current-day software development environments are inadequate to handle crosscutting concerns, making software maintenance and software evolution a difficult process.

Software classifications increase our ability to understand, modularize and evolve the software implementation by grouping together source-code entities that address a same concern. The fact that they are expressed in a (subset of) a logic language allows us to define, verify, and enforce constraints and relations among these groups of source-code entities in an intuitive way. This basic idea allows us to provide support for a whole range of tasks in the software development process: (a) provide generic and flexible code structuring mechanisms to make the software more maintainable; (b) ensure the consistent use of coding conventions throughout the software [MMW01]; (c) provide support for software evolution by explicitly codifying implicit constraints in the source code and by indicating invalidations of these constraints when the software evolves [MM02, MMW02]; (d) offer a useful abstraction for generating code that may crosscut the implementation structure [TD00]; (e) express software architectures in such a way that we can still check conformance of the source code to this architecture [MW99, M00a, M00b].

Declarative Meta Programming the UCL, UniBern and VUB partners conduct research on how the technique of Declarative Meta Programming (DMP) can be used to build state-of-the-art software development support tools. The aim is to try to express the interaction between the higher phases of software development and the implementation level in a better way. Logic meta programming is an instance of hybrid language symbiosis, merging a logic meta level language with a standard object-oriented base language.

Logic meta programming requires that the symbiosis between the logic meta language and the object-oriented base language is made explicit, allowing base level programs to be expressed as terms, facts or rules in the meta level. This can be achieved in various ways. In the case of Java, which is a statically typed language, this meta layer might be implemented as a preprocessor or even an extension of the Java compiler itself. In the case of Smalltalk, it requires the addition of a number of classes to the standard

Smalltalk hierarchy. A third approach is the use of a separate logic language and base language with provisions of the logic language to externally access the repository of the base language.

Logic meta programming is an emerging technique not quite out of the lab yet. However, it has already been shown to be very expressive. A number of successful experiments with declarative meta programming have already been carried out in the context of code generation, aspect-oriented programming, code optimization, and compliance checking between software architectures and implementation. Logic meta programming is also used to determine how inherent domain knowledge in software applications can be described in a localized and declarative manner.

Below we present a non limitative list of experiments that have been performed in the context of declarative meta programming.

• In [DV98] on Type-Oriented Logic Meta Programming, an approach is proposed to use logic meta programming as a way to extend the expressiveness of current type systems. The approach proved to be sufficiently general to handle aspect-oriented (meta)programming as well. De Volder essentially used a code generation approach where code was described at a declarative meta level including both high-level logic declarations and low-level pieces of Java source code. Based on these declarations his TyRuBa system then generated one or more Java programs satisfying these high level descriptions. Using the same TyRuBa system, a Master student conducted an experiment to generate the source code of an application by describing it at design level as a configuration of components.

• The SOUL system proposed in [W01] is a hybrid logic language implemented in Smalltalk and with a tight symbiosis with both the Smalltalk language and development environment. It seems intuitively clear that design information and architectural concerns are best codified as logic constraints or rules. Using SOUL, experiments have been carried out to check, browse for, or enforce programming conventions, design patterns and styles, and to check conformance of source code to architectural constraints.

• Despite the many advantages of adopting accepted design principles and techniques (such as idioms, programming conventions, design patterns and heuristics) when implementing software, often this results in some performance penalties. To allow for software systems with a clean design without compromising efficiency, [TD99] proposed to do source to source transformation from well-designed implementations to more efficient ones. To achieve this, he used a combination of a logic and a functional meta language. The logic language is used to declare the role certain implementation artifacts play in specific design constructs. The functional language describes the optimization transformations for each specific design construct. These transformations can rely on the information contained in the logic declarations.

• [MT01] describes some experiments to express evolution of design pattern instances at a high-level by expressing design patterns, their instances, and their typical evolutions in a declarative way. By reasoning about the software at such a high level of abstraction, evolution becomes more manageable and less error-prone.

Aspect Oriented Programming is best described by the following citation:

> *"Aspect-oriented software development is a new technology for separation of concerns (SOC) in software development. The techniques of AOSD make it possible to modularize crosscutting aspects of a system.*
>
> *Like objects, aspects may arise at any stage of the software lifecycle, including requirements specification, design, implementation, etc. Common examples of crosscutting aspects are design or architectural constraints, systemic properties or behaviors (e.g., logging and error recovery), and features.*
>
> *Researchers in AOSD are largely driven by the fundamental goal of better separation of concerns. So while crosscutting tends to be a significant focus of their work, they also incorporate other kinds of SOC techniques, including such well established approaches as OO and good old-fashioned structured programming. This is reflected in the way much of the work in the field blends support for many different kinds of modularity including block structure, object structure, inheritance as well as*

crosscutting."[13]

Several of the academic partners were involved in the advancement of this newly emerging software development discipline. In addition to fairly software-technical issues concerning persistence [MM01] it has been used as a medium in which to express architectural concerns [DVFW00, MMW00, MMW01], domain knowledge [DD99, DDGD01] and high-level code generators [DVD99]. To our knowledge, AOSD has not been investigated as a *generic* code instrumentation technique.

2.4 Anticipated foreground knowledge through the STWW project

The anticipated foreground knowledge can be summarized as follows:

- Loosely coupled architectural components
- Low-tech classification of interacting business application
- Logged behaviour as a source for reconstructing architectures
- Instrumentation of business applications using an aspect-oriented approach
- Simple ontology's to register expertise about business applications
- Aspect Oriented Software Development

For a detailed description of these items, we refer to the next sections.

[13] http://aosd.net/

3. Objectives of the project

This proposal is about the identification and application of a number of architectural resources to the integration of business software. Within the scope of this particular project, three avenues of investigation are proposed — with various levels of challenge/risk.

First of all, we propose to develop a variant of the architectural classification techniques developed by some of the academic partners to support the reengineering and evolution of software. Our ambition is to provide a lightweight framework that compresses existing data models/source code of business applications into fairly low-level architectural information providing the software architect with a first view on compatibilities and conflicts. The technology we envisage essentially boils down to code-mining the existing source code in view of partitioning it into tagged architectural entities.

Next we would like our framework to be able to express domain knowledge about the various data constituents. The object is to import design concerns/decisions, constraints, rules etc.. into the architectural model of each of the applications. To this end, we will consider a lightweight variant of a methodology developed by one of the academic partners: declarative metaprogramming. Prior experiments, using knowledge representation techniques to express rules and constraints about fairly standard software applications and their evolution, make us confident about this approach. We will endeavour to express knowledge about data (such as for instance: data ownership) using task ontology's.

Finally, we would like to investigate the use of dynamic information in the recovery of parts of the architecture of an existing business application. Whenever possible we shall consider the instrumentation of these applications; in all other cases we will look into the possibility of analyzing the external behavior such as interaction traces, logs etc. We do set high hopes in the possibility of instrumenting the source code of applications (or even available batch scripts that steer the actual applications, allowing us to view the latter as black boxes). The emerging discipline of aspect oriented software development (several of the academic partners have been more than active in this field) is an excellent approach: developing instrumentation code as an aspect will ensure a maximal independence from the code that needs to be instrumented.

Aspect oriented software development will prove to be a unifying technological and methodological platform for the entire project. It connects the three technical workpackages in a united structure.

The objective is to work in close collaboration with the industrial partners that constitute the user committee. A singular role is reserved for Inno.com (actually an academic spin-off of the 5 Flemish universities, specializing in high-level software architectures); the other industrial partners have been exposed to precisely the setting described earlier on and are particularly interested in the results of the proposed research.

To conclude: this proposal addresses an issue of particular importance to many organizations (business or otherwise); the proposed consortium has the necessary experience and expertise to investigate and produce original and relevant results. And: a channel is provided between the producers and consumers of new methods and technology to facilitate the restructuring and integration of business applications.

3.1 Scientific objectives

The major concrete and measurable scientific objectives of *ARRIBA* can be summarized as follows:

- Advancement of loosely coupled components as a flexible model for representing software architectures
- Intentional and extensional software classification as a lightweight approach to detect conflicts during the unanticipated integration of large business applications
- Instrumentation of business applications using an aspect-oriented approach

- Data/knowledge-oriented software architectures

A number of minor scientific objectives of *ARRIBA* can be summarized as follows:

- Confirmation of simple task ontology's as an approach to register expertise about data in business applications
- Using declarative metaprogramming to express conformance during unanticipated integration of business applications
- Possible new insights in using the behaviour of business applications for reconstructing architectures

3.2 Technological objectives

The major concrete and measurable technological objectives of *ARRIBA* can be summarized as follows:

- A documented architectural framework for loosely connected components
- A documented methodology for the unanticipated integration of business applications
- Prototyping intentional classification tools for business applications
- Prototyping instrumentation tools for business applications using an aspect-oriented approach

3.3 Economic objectives

ARRIBA performs innovative technological research with a long term economic effect. The economic objective of ARRIBA is to improve on theprocess of unanticipated integration of business applications. Based on the feedback from the foundation described in the beginning of this document, this is a real and widespread problem that occurs in a variety of enterprises, independent of scale and specialization of ICT related activities. A common denominator of concerned parties seems to lie with the dynamics of their organizations: frequent reorganizations (because of the nature of the enterprise; or because of the need for merging activities with similar enterprises) generate the need for integration. The economic objectives of the ARRIBA project are therefore fairly straightforward: replace an unsatisfactory process of unanticipated integration of business applications (which is generally ad hoc or at best dependent on inadequate EAI techniques) by a process that is supported by a methodology and a series of lightweight tools. The proposed package is sufficiently generic and flexible to be acceptable in a wide range of situations; it will however require a fair amount of human steering. ARRIBA should not only help a software engineer in the act of unanticipated integration; it should also provide guidelines for building software that provides hooks for automated integration tools.

3.4 Potential added value to sustainable development

Although the relationship between software development environments and sustainable development is far from obvious, in this particular case is could be argued that ARRIBA promotes the view on software as a renewable resource. In far too many cases the unanticipated integration of software results in a partial or total rewrite of the merged system. The efforts to establish maintainable and reusable single software systems (see the section on software evolution) is magnified by the ARRIBA project to cover multiple systems that need to coexist with a minimum of additional resource investment.

References

[BDMDV00] J. Brichau, W. De Meuter, and K. De Volder. Jumping aspects. In Proceedings of the ECOOP'2000 Workshop on Aspects and Dimensions of Concerns, 2000.

[BR00a] J. Brichau. Declarative composable aspects. In Proceedings of the ECOOP'2000 Workshop on Advanced Separation of Concerns, 2000.

[BR00b] J. Brichau. Declarative meta programming for a language extensibility mechanism. In Proceedings of the ECOOP'2000 Workshop on Reflection and Meta Level Architectures,

[CoCh01] Compliance Checking in Object-Oriented Systems, final technical report of a basic research program funded by the Brussels' region (available on demand), 2001.

[D01] Serge Demeyer, Sander Tichelaar and Stéphane Ducasse. FAMIX 2.1 - The FAMOOS Information Exchange Model. Tech report, University of Bern.

[D02] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Object-Oriented Reengineering Patterns. Morgan Kaufmann, 2002. to appear, spring 2002.

[D98] De Hondt, K. A Novel Approach to Architectural Recovery in Evolving Object-Oriented Systems. PhD Dissertation. Vrije Universiteit Brussel (1998).

[D99a] Serge Demeyer, Stéphane Ducasse, and Sander Tichelaar. Why unified is not universal. UML shortcomings for coping with round-trip engineering. In Bernhard Rumpe, editor, Proceedings UML'99 (The Second In-ternational Conference on The Unified Modeling Language), LNCS 1723, Kaiserslautern, Germany, October 1999. Springer-Verlag.

[D99b] Serge Demeyer, Stéphane Ducasse, and Michele Lanza. A hybrid reverse engineering platform combining metrics and program visualization. In Francoise Balmas, Mike Blaha, and Spencer Rugaber, editors, Proceedings WCRE'99 (6th Working Conference on Reverse Engineering). IEEE, October 1999.

[D99c] Serge Demeyer and Stéphane Ducasse. Metrics, Do They Really Help?. In Proceedings LMO'99 (Languages et Modèles à Objets), 1999, pages 69-82.

[DD99] M. D'Hondt and T. D'Hondt. Is domain knowledge an aspect? In Proceedings of the ECOOP99 Aspect Oriented ProgrammingWorkshop, 1999.

[DDGD01] Wolfgang De Meuter, Maja D'Hondt, Sofie Goderis, Theo D'Hondt. Reasoning with Design Knowledge for Interactively Supporting Framework Reuse, Proc. SCASE '01, 2001

[DDMW00] Theo D'Hondt, Kris De Volder, Kim Mens, Roel Wuyts. Co-Evolution of Object-Oriented Software Design and Implementation, Proc. SACT Symposium '00, Kluwer Academic Publishers.

[DDN00] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Finding Refactorings via Change Metrics, Proc. Int. Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'2000) ACM Press, 2000.

[DDW99] Maja D´Hondt, Wolfgang De Meuter, Roel Wuyts. Using Reflective Logic Programming to Describe Domain Knowledge as an Aspect, Proc. 1st Int. Symp. Generative and Component-Based Software Engineering (GCSE'99).

[Duc99] Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. A language inde-pendent approach for detecting duplicated code. In Hongji Yang and Lee White, editors, Proceedings ICSM'99 (International Conference on Software Maintenance), pages 109-118. IEEE, September 1999.

[Duc99a] Stéphane Ducasse . Evaluating Message Passing Control Techniques in Smalltalk.. Journal of Object-Oriented Programming (JOOP), SIGS Press, 12-6, pp. 39-44, 1999.

[Duc01] Stéphane Ducasse, Habilitation à diriger des recherches: Reengineering Object - Oriented Applications, sep, Université Pierre et Marie Curie (Paris 6), 2001.

[DV98] Kris De Volder. Type-Oriented Logic Meta Programming, PhD thesis, Programming Technology Lab, Vrije Universiteit Brussel, September 1998.

[DV98] K. De Volder. Type-Oriented Logic Meta Programming. PhD thesis, Vrije Universiteit Brussel, 1998.

[DVD99] Kris De Volder, Theo D'Hondt. Aspect-Oriented Logic Meta Programming, Proc. 2nd Int. Conf. Meta-Level Architectures and Reflection, LNCS 1616, pp. 250-272, Springer-Verlag, 1999.

[DVFW00] K. De Volder, J. Fabry, and R. Wuyts. Logic meta components as a generic component model. In Proceedings of the ECOOP'2000: Fifth International Workshop on Component-Oriented Programming, 2000.

[DW00] D. Deridder and B. Wouters. The use of an ontology to support a coupling between software models and implementation. In Proceedings of the ECOOP'2000 Workshop on InternationalWorkshop on Model Engineering, 2000.

[J96] Dirk Janssens and Tom Mens, "Abstract Semantics for ESM Systems", Fundamenta Informaticae, 26, 315-339, 1996.

[J99] Dirk Janssens, "Actor Grammars and Local Actions" in "Handbook of Graph Grammars and Computing by Graph Transformation", vol 3, World Scientific, pp. 57-106, 1999.

[KLM+97] G. Kiczales, J. Lamping, A.Mendhekar, C.Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In Proceedings of ECOOP'97, pp. 220–242. Springer Verlag, 1997. LNCS 1241.

[L97] Lucas, C. Documenting Reuse and Evolution with Reuse Contracts. PhD Dissertation. Vrije Universiteit Brussel (1997).

[Lanz01] Michele Lanza and Stéphane Ducasse. A Categorization of Classes based on the Visualization of their Internal Structure: the Class Blueprint. In Proceedings of OOPSLA 2001, 2001.

[M00a] K. Mens. Automating Architectural Conformance Checking by means of Logic Meta Programming. PhD thesis, Department of Computer Science, Vrije Universiteit Brussel, Belgium, 2000.

[M00b] K. Mens. Multiple cross-cutting architectural views. Position paper, Second Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (ICSE 2000).

[MM02a] Kim Mens, Tom Mens. Software evolution through intentional classifications. Extended abstract. Network meeting Foundations of Software Evolution, 2002.

[MMW00] T. Mens, K. Mens, and R. Wuyts. On the use of declarative meta programming for managing architectural software evolution. In Proceedings of the ECOOP'2000Workshop on Object-Oriented Architectural Evolution, June 2000.

[MMW01] K. Mens, I. Michiels, and R. Wuyts. Supporting software development through declaratively codified programming patterns. In Proc. Software Engineering and Knowledge Engineering, pages 236-243. Knowledge Systems Institute, 2001.

[MMW02] Kim Mens, Tom Mens, Michel Wermelinger. Supporting software evolution with intentional software views. Short paper submitted for Int. Workshop Principles of Software Evolution, 2002.

[MN95] G. Murphy and D. Notkin. Lightweight source model extraction. In Proceedings of SIGSOFT'95, Third ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 116–127. ACM Press, 1995.

[MNS95] G. Murphy, D. Notkin, and K. Sullivan. Software reflexion models: Bridging the gap between source and high-level models. In Proceedings of SIGSOFT'95, Third ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 18–28. ACM Press, 1995.

[MT01] Tom Mens and Tom TourwÈ. A Declarative Evolution Framework for Object-Oriented Design Patterns. Proc. Int. Conf. Software Maintenance, pp. 570-579, IEEE Computer Society Press, 2001.

[MWD99] K. Mens, R. Wuyts, and T. D'Hondt. Declaratively codifying software architectures using virtual software classifications. In Proceedings of TOOLS-Europe 99, pp. 33–45, June 1999.

[Rich98a] Tamar Richner. Describing Framework Architectures: more than Design Patterns. Proceedings of the ECOOP '98 Workshop on Object-Oriented Software Architectures, July, 1998.

[Rich99a] Tamar Richner and Stéphane Ducasse. Recovering High-Level Views of Object-Oriented Applications from Static and Dynamic Information. In Proceedings ICSM'99 (International Conference on Software Maintenance), IEEE, pp. 13-22, 1999.

[THS99] P. Tarr, H. Ossher, W. Harrison, and Jr. S. M. Sutton. N degrees of separation: Multi-dimensional separation of concerns. In International Conference on Software Engineering (ICSE 1999), 1999.

[SH01] Michael Stonebraker , Joseph M. Hellerstein, Content integration for e-business ACM SIGMOD Record, Proceedings of the 2001 ACM SIGMOD international conference on Management of Data on Management of data May 2001

[SLMD96] Steyaert, P., Lucas, C., Mens, K. and D'Hondt, T. Reuse Contracts - Managing the Evolution of Reusable Assets. Proceedings of OOPSLA '96, SIGPLAN Notices, 31(10). ACM Press (1996) 268-286.

[SoFa02] Software Factories, Final technical report for an IWT basic research program (draft text available), 2002.

[T01] Sander Tichelaar, "Modeling Object-Oriented Software for Reverse Engineering and Refactoring," Ph.D. thesis, University of Berne, December 2001.

[TD00] T. Tourwe and K. De Volder. Using software classifications to drive code generation. Position paper, ECOOP 2000 Workshop on Objects and Classification: a Natural Convergence.

[TD99] T. Tourwé and W. De Meuter. Optimizing object-oriented languages through architectural transformations. In 8th International Conference on Compiler Construction, pp. 244–258, 1999.

[Tich00b] Sander Tichelaar, Stéphane Ducasse, Serge Demeyer and Oscar Nierstrasz. A Meta-model for Language-Independent Refactoring. In Proceedings ISPSE 2000, IEEE, pp. 157-167, 2000.

[VD00] Werner Van Belle, Theo D'Hondt: Agent Mobility and Reification of Computational State, an experiment in migration, Structure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, Springer Verlag 2000.

[VVD99] Werner Van Belle, Karsten Verelst, Theo D'Hondt: Location Transparant Routing in Mobile Agent Systems Merging Name Lookups with Routing, Future Trends of Distributed Computer Systems 1999

[W01] Roel Wuyts. A Logic Meta-Programming Approach to Support the Co-Evolution of Object-Oriented Design and Implementation, Phd Thesis, Programming Technology Lab, Vrije Universiteit Brussel, January 2001.

[WDVP00] B. Wouters, D. Deridder, and E. Van Paesschen. The use of ontologies as a backbone for use case management. In Proceedings of the ECOOP'2000 Workshop on Objects and Classifications, a natural convergence, 2000.

[WOSEF00] WoSEF: Workshop on Standard Exchange Format; Susan Elliott Sim, Rainer Koschke;Software Engineering Notes Vol26(1), January 2001, pp. 44-60, ACM Press.