K-Mediator: Towards Evolving Information Systems

Hussein Zedan, Shikun Zhou, Nilesh M Sampat, Xiushan Chen, Antonio Cau
Software Technology Research Laboratory
SERCentre, De Montfort University,
The Gateway, Leicester LE1 9BH, England
Email: {hzedan,kzhou,nmsampat,xiu,acau}@dmu.ac.uk

Abstract

Business processes and goals change rapidly due to the turbulent environment in which they operate. Their supporting and underpinning technologies may, as a result, require to change. Meanwhile, technological advances are being made at an increasing rate. This may trigger changes in business goals and processes to exploit these new advances; a notable example is *e-commerce* and *e-learning*. Understanding and analysing the interplay and the dual effect between these two entities, technologies and business, is vital both for the prosperity of business and the success and further development of the technologies themselves. This paper proposes the K-Mediator framework together with its underpinning theory to facilitate the co-evolution between businesses and their supporting technologies.

1 Introduction

Business environment, both at global and local levels, is changing rapidly. Multinational corporations are facing growing challenges caused by, e.g., the changes in the world monetary systems, competitive mergers and national governments striving towards accelerated economic progress. These shifts in the market often lead to the need for redefining business *goals* which subsequently affect the requirements of the underpinning technologies (e.g., computing systems). Even the installation of new technology affects business *practices* and hence leads to changes in such requirements. System *evolution* is therefore often attributed to changes in system's requirement. In fact, the inability to deliver all business required functions, results in incremental development that also lead to system evolution.

Meanwhile the underpinning technology is also changing rapidly, and the business may need to change to exploit this technology (e.g. e-commerce and e-learning). The inter-relationships between the changing business environment, its impact on the requirements of the end-user and business software applications are important. Fundamental issues include:

- Understanding the nature and meaning of evolution within an organisational context and of co-evolution of related socio-technical entities within a social ecosystem and

- Identifying, articulating and analysing (i) the conditions which enable and constrain co-evolution, and (ii) some specific instances of the phenomenon.

Some key questions may include what are the socio-technical conditions, which enable and constrain co-evolution between organisational change and software evolution? Do changes in these conditions affect the rate of co-evolution? What necessitates the (re)configuration of software components? What new organisational procedures or patterns may emerge from the reconfiguration of software components using a domain specific language? Can the resulting end-user developed system exhibit emergent behaviour, which influences organisational environment and direction? To what extent (if at all) is this behaviour predictable? Can the network of decisions and interactions, that lead to the development or reconfiguration of an information system, be traced? How does the end-user select which components/domain concepts to use in building a system? How are end-user selections influenced by the organisational dynamics? What is the role of feedback in fixing end-user choices within the organisation, amplifying their effect and establishing technical platforms, component usage and domain languages?

In this paper we present the **K-Mediator** ('K' stands for 'Knowledge') framework, within which the notion of co-evolution may be studied and analysed in the setting of Information System (IS) design and development. The K-Mediator framework could be viewed from different levels of abstractions. At a conceptual level, the framework provides the role of an architect in the procurement process which results in making decisions at an early stage of software development life cycle. At a concrete level, the framework provides a set of *business-specific* components from which a system may be constructed and configured/re-configured rapidly and efficiently from new or existing business components. Three main attributes must therefore be satisfied for such component-based development infrastructure. Components must be able to be *plugged* together easily. *Agility* is another important characteristic for it permits rapid modifiability. For business-critical applications, *high-integrity* is fundamental as it ensures high level of assurance through 'provably correct' components.

A key advantage of such a rapid assembly is that the assembler, who is also playing the role of a 'mediator', has a *clear* understanding of the business goals. Hence the constructed system is guaranteed to be driven by the business need; a requirement that is crucial for the success of the system. Additionally, a gained advantage is that system requirements need not to be captured (or negotiated) as the 'mediator' can freely *articulate* system's intentions through components selection without the intervention of a technology expert. This view is predicated on the need for fast product development, turbulence in the business environment, semantic inoperability, the individual nature of requirements and the limited availability of IT resources due to downsizing and outsourcing.

Paper Organisation

The paper is organised as follows. Section 2 provides a rationale for the K-Mediator and also gives a taxonomy of change at both business and technologies. It articulates each of these changes and present the case for its effect. Section 3 presents the K-Mediator and its roles. In Section 4, we present our component-based development method. and conclude, in Section 5,

with a discussion and future and related work.

2 Rationale and Taxonomy for Change

In this section we articulate the problems often found in building a software solution that meets business needs and goals, setting the scene for a *business-driven* solution in which evolution at the business side could be rapidly realised. Furthermore, we provide a small taxonomy of changes from both the business and technology viewpoint.

2.1 Business Needs and IT Solutions

Business needs arise from the inescapable factors in any business's external environment. Most obviously they can be competitors, and therefore part of the market, not only competing with their outputs but even for inputs including employee skills, raw materials and components. Business needs can be at various levels:

Operational: E.g. to discover the address of a supplier

Tactical: E.g. to decide on the most relevant supplier

Strategic: E.g. to decide which firm to try to buy up in order to guarantee the supply of an essential component.

Similar information may be used for different purposes. For an operational purpose, it may provide a definite answer; for strategic purposes, it may only be one factor to reduce the risk in a decision.

Business needs are often realised by a solution in software:

$$\text{Needs} \Rightarrow \text{Solution}.$$

However, in practice, the transition from identifying the need (which is a large problem in its own right) to building a solution is divided into two parts [13]. The need leads to a specification of a solution and then a solution is built that conforms to the specification. Thus

$$\text{Needs} \Rightarrow \text{Solution}$$

becomes

$$\text{Needs} \Rightarrow \text{Specification} \qquad \text{and} \qquad \text{Specification} \Rightarrow \text{Solution}.$$

Each part of such a process invariably involves a domain expert whose expertise and knowledge does not often cross other domains. This results in the creation of many gaps that need to be bridged if we are to be *certain* that the resulting solution indeed *meets*[1] the original business needs; a fundamental issue to the whole process which has not yet been fully resolved.

[1] Known as *satisfiability* problem

In fact, involving many domain experts may be seen as a major reason for the creation of these gaps. Additionally, it makes the whole process time-consuming (undesirable aspect especially in a rapidly changing and volatile business environment) and hence expensive and prone to errors. Economics, and other factors, therefore dictate the need to seek a faster process for realising the business needs.

The K-Mediator framework plays an important role in achieving such a goal. From identifying business *needs/goals* to the *decision making process* for selecting the appropriate software solution through *traceability* mechanisms which ensures that needs are traced to a solution. The K-Mediator may thus be seen as a 'conceptual' and 'technical' middleware that resides between business goals and their realisation.

2.2 Taxonomy of Change

We believe that a well defined taxonomy of changes will play a key role when people identify problems and endeavour to find best strategic solutions, in both business and technical worlds. There are two major classes of changes depending on their trigger.

2.2.1 Changes Triggered by the Business World

The following are main classes of triggers that affect and lead to changes in the business world. It should be noted that these are loosely coupled triggers and in some organisation are indeed inter-dependent.

Politics

Allocation of power and resource are two main factors that trigger changes. Tichy [17] summarised the main issues of political entities. They are *Succession Concerns*, which refer to managing structure of the organisation; *Goal Concerns*, which is the result of deciding and lengthy political-bargaining process for various organisation's goals; *Means of Doing the Work Concerns*, which is caused by different strategies of how best to carry out the organisation's goals; *Environmental Changes*, and *Reward Reallocations* which concerns on how to satisfy all competing coalitions of the organisation.

A general example is that both government rhetoric and action has created a context in which an organisation has been more prepared to undertake certain fundamental changes (e.g., a medical centre may have to undertake a new goal of teaching instead of pure health care, under the governmental power to push them to do so).

Organisation

Changes in this system come from reformation or reorganisation of business organisations themselves, such as merging/splitting organisations and opening/closing coalitions within organisations. One recent example is that the UK's largest company, Vodafone, was battling for control of German telecoms group Mannesmann. Another was the turmoil in the financial sector when Bank of Scotland and Royal Bank of Scotland were fighting over NatWest [12].

Economy

Economics changes are especially caused by events in economic circumstances from which they emerge. For example, the world recession of 1979-1981, triggered by the rise in oil prices following the outbreak of the Gulf War and deepened by the tightening of monetary controls in the Western economies, formed a painful break-point for most publishers[14]. Some publishers have been forced to widen their competitive base, like expanding from Europe to the Far East, and to combine growth (a doubling their business) and internationalism.

Technology

Technical changes alter the information-processing capacity of an organisation. It involves adjusting components of the organisational model. These adjustments are made either to increase or decrease the organisation's capacity to handle different goals. Technical alignment in the business world is designed to manage the development of the organisation and uncertainty of environment with tight regards to the financial and business criteria outputs. For example, people can be prescribed to interact with others via formal committees, task forces, etc., in order to increase information-processing capacity. At Volvo, each work group assembles a total engine rather than one piece on a long assembly line.

2.2.2 Changes Triggered by Technical Requirements

Performance

Performance plays an important part as a trigger for technological changes. The cost of high-performance hardware is continually declining which results in the demand for upgrading the current system. As an organisation moves to a high-performance configuration, business processes will need to change as business operations can now be performed with higher speed. This in turns requires business process re-engineering.

Computing Paradigms and Versions

Advances in computational paradigms have also an impact in organisations. This can be seen, for example in the ever increasing requests for system migration from procedural to object-oriented languages. This is perceived to have benefit to the organisation from a reusability point of view.

A similar situation occurs when the deployment of a new version of software takes place. For example, changing from Window 95 to Window 2000 or Window NT can have an impact on the way people within an organisation work.

New Emergent Technologies

New technologies are perhaps the fundamental trigger within the technological infrastructure of an organisation. The Internet is the most celebrated example of this situation.

3 K-Mediator

As we mentioned earlier, the K-Mediator framework may be viewed from various levels of abstractions. In this paper, we discuss the framework from a conceptual viewpoint and its underlying supporting environment.

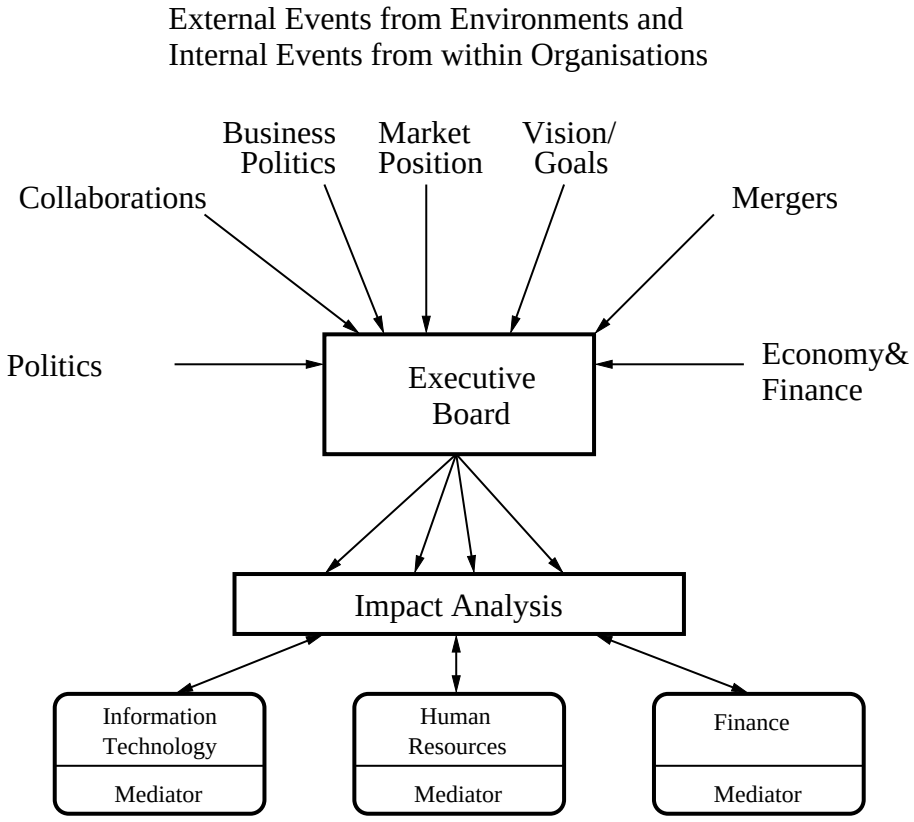

Figure 1: K-Mediator

3.1 K-Mediator: Actor and Role

A K-Mediator as an actor has the role of mediating between the business and technology domains. Therefore, a K-mediator has an inside knowledge of both worlds: business needs and goals, and the software assets available within the organisation (this is depicted in Figure 1). Fundamental to its role (which distinguishes it from, for example, an architect, analyst, requirement engineer, or domain engineer) is the immediate (first-hand) access to knowledge of the business needs that requires a software solution. Such first-hand knowledge reduces errors that will inevitably occur when articulating requirements, for example as done by an analyst or a requirement engineer. In addition, it ensures 'real' participation in system's development and hence securing its ownership and responsibility.

In this way, the K-mediator role may be described as a *business champion*. His sole responsibility is to generate innovative ideas by utilising existing resources and visualise/(re-)configure software applications from those resources.

Within the E-business world, time is very much critical, i.e., systems and their development must conforms to tight deadlines. Innovative ideas in this competitive E-business arena are emerging constantly which give a K-mediator a *reactive* role.

A K-mediator role is fundamental in shaping the way technology and business goals evolve and for controlling their impact. This is justified by the advent of new technologies such as Component Based Development and the E-Business revolution, where business needs are taking a lead role.

3.2 K-Mediator: Supporting Infrastructure

Recent interests in component technology will indeed enable the K-Mediator to realise its roles efficiently and rapidlly. The advantage of mapping a given interface to services offered by a component is that requirements can be mapped once we know what information (interface) is required from a service. For example, a Bank may have a 'vision' to provide financial solutions to its personal and business customers. To achieve such a vision there are various needs/requirements that a bank needs to fulfill, e.g., provide banking solutions to both business and personal customers. There are various *projects* that banks must undertake in order to fulfill the bank's need and one of those project is the ATM system. The ATM system provides their customer with several features for example withdrawal, balance enquiry, etc. Each of these features carries out its intended goal, i.e., to dispense money when customers present themselves with valid id (PIN) and request a withdrawal of a particular amount. The ATM system acts as a Interface between the customer and the bank. It accepts details from the customer and calls upon the services of the appropriate component. Services such as *withdraw(account, money)* or *validateCustomer(AcNo, PIN)* (see Figure 2).

3.3 K-Mediator: Decision Making

The decision making process by the K-mediator can be divided into two phases. The first phase is where decisions are made based on the business needs that map to the specification and the second phase is to make decisions to build software from the specification.

The kind of information which is available to the K-Mediator for the first phase can be business vision, business needs, number of projects that satisfy the business needs, the objectives of each project.

Information that is helpful and can be used for decision making process at phase two can include the following:

At Organisational Level: Total number of business needs, total number of projects, etc.

At Business Needs Level: Total number of projects per need, etc.

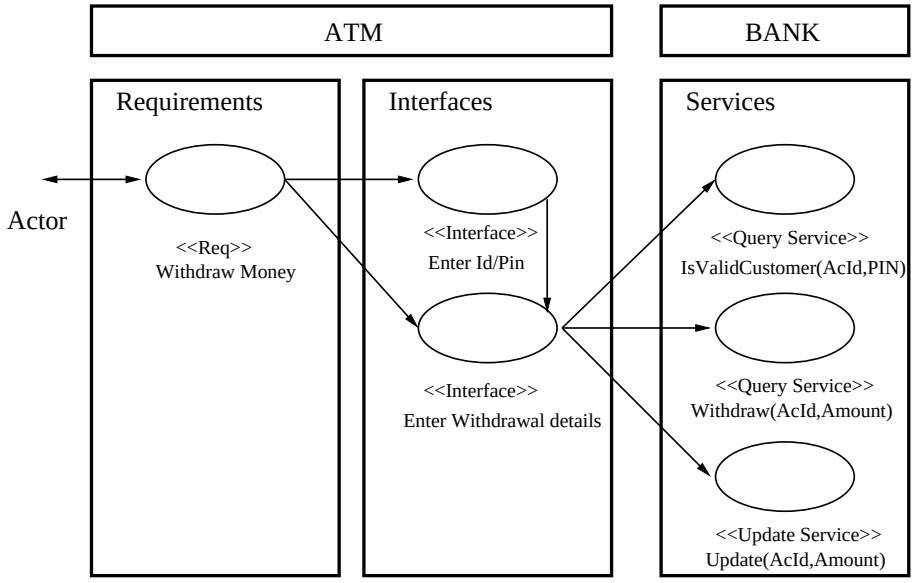

Figure 2: ATM System

At Project Level: Number of business needs that are satisfied by a project, number of goals per project, number of use cases in a project, number of interfaces in a project, number of components used in a project, number of tasks in a project, etc.

At Use Case Level: Number of actors for each use case, number of scenarios for each use case, number of main scenarios for each use case, number of extension scenarios for each use case, number of interfaces for use cases, etc.

At Interface Level: Number of goals that are supported by each interface, number of functionalities that are supported by each interface, number of services that are required by each interface, etc.

At Service Level: Number of parameters per service, etc.

At Component Level: Number of services offered by a component, etc.

4 Agile Components

In this section, we elaborate on the technological support for the K-mediator. In particular we present a *component-based* model of computation which provides the backbone of our framework. To fulfill the roles of the K-Mediator, three main characteristics for a component must be ensured:

1. *Plugability.* Mechanisms must exist to allow components to be 'easily' plugged (composed) together to form a system.

2. *Agility*. This is a property that provides more flexibility to components. This may be materialised in our ability to 'modify' components.

3. *High Integrity.* High integrity components are essential for the successful operation of a business or enterprise in business critical systems. High levels of assurance are desirable and are achieved through formal verification.

It should be noted that plugability ensures system's extensibility whilst agility could be utilised by providing a repository of generic system components which can be instantiated to meet specific requirements. High integrity components implies that our component-based model must be formalised using sound and precise formal notations.

4.1 Development Framework

The overall sketch of the proposed development framework is depicted in Figure 3.

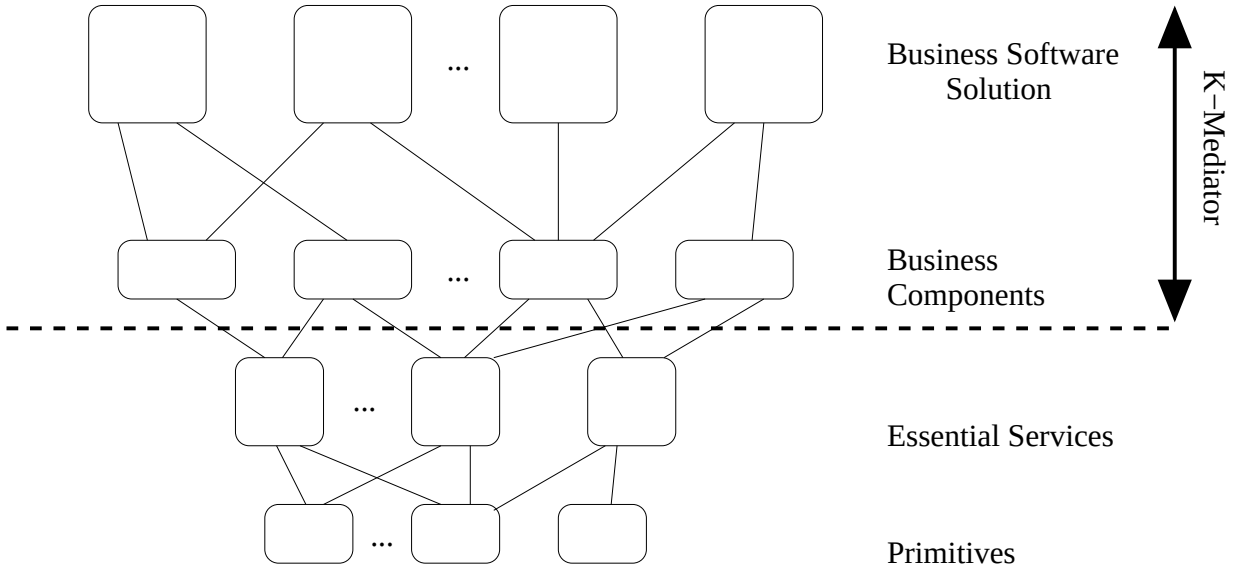

Figure 3: Development Framework

A fundamental aspect in our framework is the provision of Essential Services (ES). These are seen as primitive services to all (kinds of) businesses; they form a generic service repository. Each service in this layer is closely related to an elementary business task that has been specified through domain analysis and modelling. Whilst the Business Components (BC) are constructed using a (rich) set of Business Specific Languages (BSLs). Each construct in a BSL is being supported or implemented by one or more of the Essential Service(s). The construction of the Business Components is performed through the grammar provided by the specific BSL. For example in a financial business we may have *Account, Customer, Transaction, Open, Withdraw* and *Money* to be part of the financial BSL. An Essential Business Component may be constructed

as *OpenCustomerAccount* or *WithdrawMoney from CustomerAccount* where 'concatenation' and *from* are part of the Business Specific Language grammar.

The *Primitives* are the usual computational primitive constructs: Assignment, Input/Output, and Null. Primitives are the basic building blocks for the construction of Essential Services.

It should be noted that Business Software Solution are constructed using only the BCs. BCs are relevant to the particular business under consideration. These may differ from one business to another within the same domain. BCs reflect the business culture (including the current process) and vocabularies used.

4.2 Components, Services and Features

The unit of computation, at the K-Mediator level in our framework, is component. A component has a set of interfaces which are the 'gateways' through which other components can access its services. Each service has a distinct set of *features*. In our model we define features to be properties of the service. For example if PRINT is a component's service, then Duplex, Simplex, PageType and PageRange are features of the service. A component take the general form

```
COMPONENT CName
    PROVIDEDSERVICES:
        [pr1] A: (f11,...,f1n; f21,...,f2n;...)
        [pr2] B: (g11,...,g1n; g21,...,g2n;...)
        .
        .
        .
    REQUIREDSERVICES:
        C: (h11,...,h1n; h21,...,h2n;...)
        D: (i11,...,i1n; i21,...,i2n;...)
        .
        .
        .
END CName
```

As can be seen, a component is a black-box *encapsulating* services. It is identified by its name (CName) and is accessed by its *interfaces*. There are two types of interfaces: *ProvidedServices* and *RequiredServices*. The former is a list of services that are provided by the component. Each Service has the usual input and output parameters (for the sake of exposure simplicity, these are not dealt with here). Fundamental to a service is the provision of a family of features in which each member is a set of related features (e.g. fk1,...,fkn). The provided services are prioritised. Services priority is optional, in which case all provided services have equal priority. Priorities are partially ordered where 1 is the highest. *RequiredServices* are those services from other components (in the system or repository) that the component may need to perform its services. This provides a powerful mechanism for adding or replacing features of a component; hence increasing the component's agility.

Components interact with each other and the outside world through service calls. Each call specifies the requested service(s) together with their chosen features. For example, the following component provides printing and scanning services each with various features:

```
COMPONENT PrintScan
    PROVIDEDSERVICES:
      [1] print: (Simplex, Duplex; A4, A3, letter; #n)
      [2] scan : (HighResolution, LowResolution)
    REQUIREDSERVICES:
      Filter.Transform: (TextToPostscript, PostscriptToText)
      .
      .
      .
END PrintScan
```

Here, the 'print' service is of higher priority than the 'scan' service. There are three 'classes' of features for the print service: type of printing, papers and number of copies. The scanning service provides two features: either High or Low resolution. The ';' separates the classes of features. Features in one class are mutually exclusive.

PRINTSCAN also requires various services from other components. For example, the Transform service from the FILTER component with its features TextToPostscript, PostscriptToText. PRINTSCAN may be called using, for example

```
PrintScan.print: Duplex
```

This activate the service print with the duplex feature. By convention, the print will be performed on A4 paper and only one copy. However

```
PrintScan.print: Duplex; letter; 4
```

will result in 4 copies printed on letter paper and in a duplex mode.

However, a service may also be called without specifying a particular feature. In such a case, a *default* feature will be chosen. By convention, the first in the list of features is taken. For example

```
PrintScan.print:
```

will print in simplex form. To add features within a service, we create a new service with the same name whose feature(s) are those that we require. For example, if we wish to add the feature Booklet to the print service, PrintScan is modified as follows

```
COMPONENT PrintScan
    PROVIDEDSERVICES:
      [1] print: (Simplex, Duplex; A4, A3, letter; #n)
```

```
        [2] scan : (HighResolution, LowResolution)
        [1] print: (Booklet; ; )
    REQUIREDSERVICES:
        Filter.Transform: (TextToPostscript, PostscriptToText)
        .
        .
        .
END PrintScan
```

It should be noted that the above is semantically equivalent to

```
COMPONENT PrintScan
    PROVIDEDSERVICES:
        [1] print: (Simplex, Duplex, Booklet; A4, A3, letter; #n)
        [2] scan : (HighResolution, LowResolution)
    REQUIREDSERVICES:
        Filter.Transform: (TextToPostscript, PostscriptToText)
        .
        .
        .
END PrintScan
```

This is because both representation have the same family of features. However, at the implementation level, they are different as the former realises the features in two different services.

Notice that all services in the repository are *reactive* awaiting to provide their services. This reflects the reactiveness of system.

A system may be constructed and/or (re)configured by the assembly of the various services packaged within the components. The chosen services are composed together within the assembly using the standard composition operations: sequential (;), iteration (WHILE <boolean> DO), conditional (IF <boolean> THEN <...> ELSE), etc.

```
SYSTEM Sys_Id
    Body ::= service ; service |
             WHILE <boolean> DO service |
             IF <boolean> THEN service ELSE service |
             service || service
END Sys_Id
```

The high integrity of a component is reflected in way the services are specified and verified. In [18, 2] we have provided a formal model based on Interval Temporal Logic (ITL) [10, 11, 1]. Such a model is suitable for providing compositional formal semantics to the services (and hence components.) The compositional proof system of the ITL framework is also suitable for verification and formal analysis.

5 Concluding Remarks

Capturing, or *negotiating*, requirements is hard. There have been various approaches towards such a goal from structured interviews and analysis to ethnography. The most notable of them is *use-case* [9] introduced by Jacobson. It defined use case as a sequence of transactions initiated by a user of the system (an *actor*) and could be realised by *sequence* or *collaboration* graphs. The technique has also been used to capture business processes within the domain of Business Process Engineering [7]. Various difficulties have been identified with the use cases technique. These include: how to write use cases, what to include in use cases, how to structure use cases, how big the use case should be, the levels at which use cases should be applied, how to control scenario explosion, and what shall be done with use case once they become outdated.

These drawbacks and limitations were addressed in various use case-related approaches [3, 4, 5, 15, 8, 6] which addressed the need to map business needs to system functionalities. In [16] we studied and compared representatives of these approaches. None of the approaches provides a full solution to the problem. However, an integration between *Use case with goals* (UCWG) [3] and *Requirement, Services and Interface* (RSI) [4] approaches can provide a solution that maps from business needs to services offered by the software components.

We believe that the origin of the gap between business objectives and its underlying technologies stems from the language barrier between the two layers. On the one hand, failure to adequately capture business requirements due to, for example lack of articulation from the business side, and on the other, deficiency in understanding business goals and objectives by the champion of technologists. This gap gets wider as a result of continual *evolution* of both levels. What we require is a mechanism by which such gap may be *bridged*. In this chapter, we have outlined a framework, **K-Mediator**, that alleviate this problem. A key characteristic of the approach is that we give the responsibilities of constructing and (re)configuring systems to those who have adequate business knowledge as they are able to articulate business goals and objectives rather than relaying it to a requirement/design engineer.

In addition to system's construction, the K-Mediator plays a decision making role at an organisational, business and project levels.

Crucial to the framework is the provision of a development infrastructure 3 which is a novel approach of computation. This is based on **services** and **features** packaged within **components**. The infrastructure enjoys *plugability* (through 'provided' and 'required' services), *agility* (using features embedding) and *high integrity* via formal verification of correctness. Services are put in an organisational-wide repository and continually updated as a result of either IT-evolution or business needs.

We are currently in the process of building a tool support for the K-Mediator. This tool has the following components

1. A graphical Business Service Language for the construction and the assembly of components and systems;

2. A decision making facilities that enables the implementation of business goals and objectives; and

3. Traceability mechanism that ensures that business needs are traced back to systems and that the developed system satisfies business needs. In Figure 5, we show such a facility.

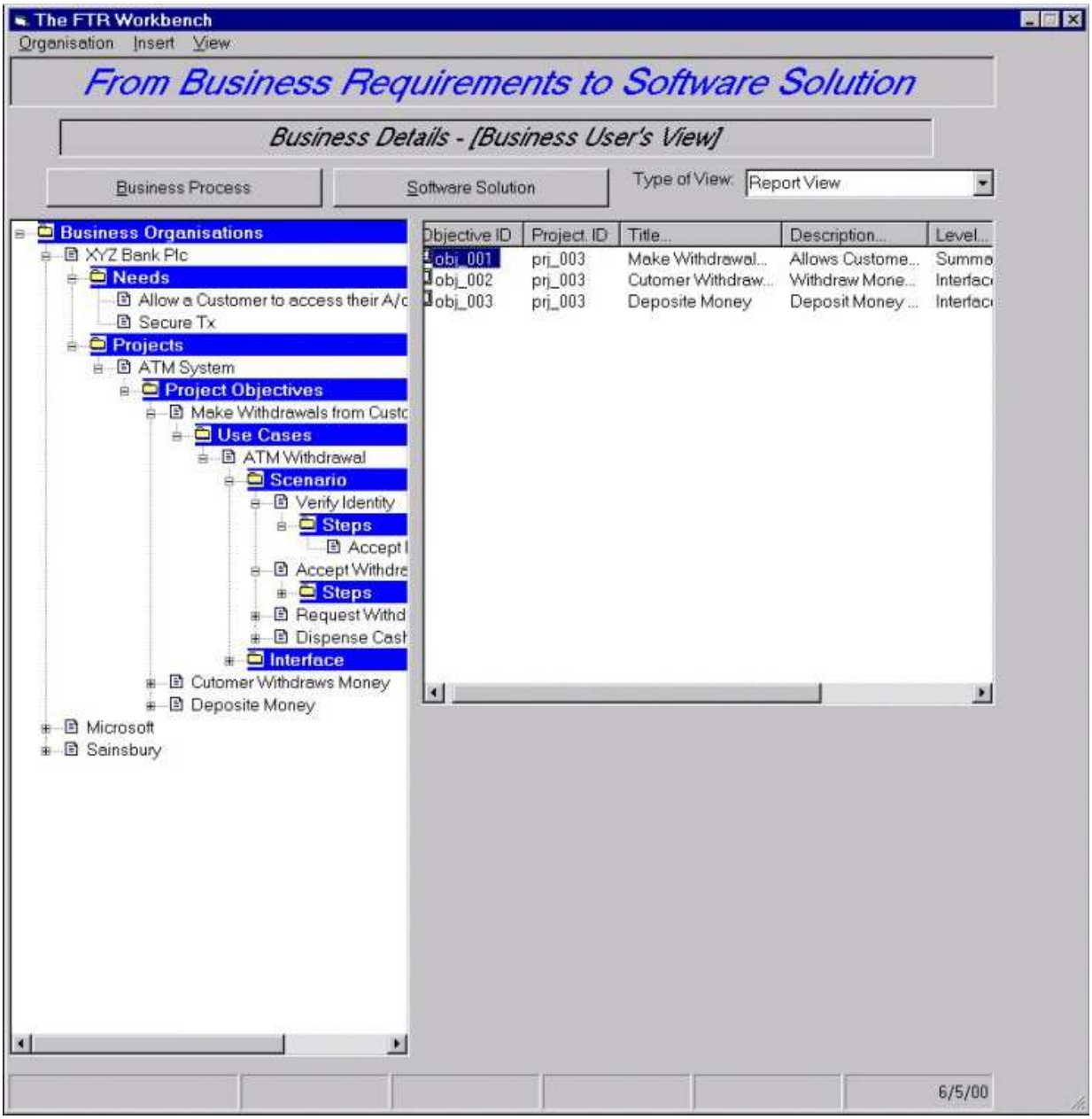

Figure 4: Traceability

References

[1] A. Cau and H. Zedan. Refining Interval Temporal Logic Specifications. In M. Bertran and T. Rus, editors, *Transformation-Based Reactive Systems Development*, number 1231 in LNCS, pages 79–94. AMAST, Springer-Verlag, 1997.

[2] A. Cau and H. Zedan. The Systematic Construction of Information Systems. In Peter Henderson, editor, *Systems Engineering for Business Process Change*, chapter 21, pages 264–278. Springer Verlag, 2000.

[3] A. Cockburn. Structuring Use Cases with Goals. *Object Oriented Programming*, Sept-Oct and Nov-Dec 1997.

[4] M. Collins-Cope. The 'RSI' Approach to Use Case Analysis: A pattern for structured use case development. *C++ Report*, 11(7), July/August 1999.

[5] L. L Constantine and L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley, Reading, Massachusetts 01867, 1999.

[6] I. Graham. *Requirements Engineering and Rapid Development: A Rigorous, Object-Oriented Approach*. Addison-Wesley, Edinburgh Gate, Harlow, Essex CM20 2JE, England, 1998.

[7] I. Jacobson. *The Object Advantage - Business Process Re-engineering with Object Technology*. Addison-Wesley, Menlo Park, CA, 1994.

[8] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1998.

[9] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Reading, Massachusetts 01867, 1992.

[10] B. Moszkowski. *A Temporal Logic for Multilevel Reasoning about Hardware*. IEEE Computer Society, February 1985.

[11] B. Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, Cambridge UK, 1986.

[12] BBC News. The Insatiable Merger Appetite. http://news.bbc.co.uk/hi/english/business/newsid_576000/576197.stm, Dec 1999.

[13] A. O'Callaghan. Getting What you Want. *Sigs Application Development Advisor*, 3(5):64–67, May/June 2000.

[14] A. Pettigrew and R. Whipp. *Managing Change for Competitive Success*. Blackwell Publishers Ltd, 1991.

[15] D. Rosenberg and K. Scott. *Use Case Driven Object Modeling with UML: A Practical Approach*. Addison-Wesley, Reading, Massachusetts 01867, March 1999.

[16] N. M. Sampat, A. O'Callaghan, and H. Zedan. From Business Needs to Software Solutions: Comparing Use Case Driven Approaches for Component Based Development. In *The Fifth International Conference on Computer Science and Informatics (CS&I) 2000*, Trump Taj Mahal Casino and Resort, Atlantic City, NJ, USA, 27 Feb - 3 Mar 2000.

[17] M. Tichy. *Managing Strategic Change: Technical, Political, and Cultural Dynamics*. John Wiley & Sons, Inc, 1983.

[18] H. Zedan, A. Cau, and B.C. Moszkowski. Compositional Modelling: The Formal Perspective. In David Bustard, editor, *Systems Modelling for Business Process Improvement*, chapter 21, pages 333–354. Artech House, 2000.