

Middlesex University Research Repository

An open access repository of

Middlesex University research

<http://eprints.mdx.ac.uk>

Clark, Tony, Barn, Balbir ORCID logo ORCID: <https://orcid.org/0000-0002-7251-5033> and
Oussena, Samia (2012) Simulating enterprise architecture models. *Infosys Journal of
Research*, 10 (3) . pp. 3-20. [Article]

Final accepted version (with author's formatting)

This version is available at: <https://eprints.mdx.ac.uk/9562/>

Copyright:

Middlesex University Research Repository makes the University's research available electronically.

Copyright and moral rights to this work are retained by the author and/or other copyright owners unless otherwise stated. The work is supplied on the understanding that any use for commercial gain is strictly forbidden. A copy may be downloaded for personal, non-commercial, research or study without prior permission and without charge.

Works, including theses and research projects, may not be reproduced in any format or medium, or extensive quotations taken from them, or their content changed in any way, without first obtaining permission in writing from the copyright holder(s). They may not be sold or exploited commercially in any format or medium without the prior written permission of the copyright holder(s).

Full bibliographic details must be given when referring to, or quoting from full items including the author's name, the title of the work, publication details where relevant (place, publisher, date), pagination, and for theses or dissertations the awarding institution, the degree type awarded, and the date of the award.

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Middlesex University via the following email address:

eprints@mdx.ac.uk

The item will be removed from the repository while any claim is being investigated.

See also repository copyright: re-use policy: <http://eprints.mdx.ac.uk/policies.html#copy>

Middlesex University Research Repository:

an open access repository of
Middlesex University research

<http://eprints.mdx.ac.uk>

Clark, Tony; Barn, Balbir; Oussena, Samia, 2012. Simulating enterprise architecture models. Available from Middlesex University's Research Repository.

Copyright:

Middlesex University Research Repository makes the University's research available electronically.

Copyright and moral rights to this work are retained by the author and/or other copyright owners. No part of the work may be sold or exploited commercially in any format or medium without the prior written permission of the copyright holder(s). A copy may be downloaded for personal, non-commercial, research or study without prior permission and without charge. Any use of the work for private study or research must be properly acknowledged with reference to the work's full bibliographic details.

This work may not be reproduced in any format or medium, or extensive quotations taken from it, or its content changed in any way, without first obtaining permission in writing from the copyright holder(s).

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Middlesex University via the following email address:
eprints@mdx.ac.uk

The item will be removed from the repository while any claim is being investigated.

Simulating Enterprise Architecture Models

By Balbir S. Barn, Tony Clark and Samia Oussena

Business and ICT strategic alignment remains an ongoing challenge facing organizations as they react to changing requirements by adapting or introducing new technologies to existing infrastructure. Enterprise Architecture (EA) has increasingly become relevant to these demands and as a consequence numerous methods and frameworks for pursuing EA have emerged. However these approaches remain bloated, time-consuming and lacking in precision. This paper proposes a lightweight method for EA (LEAP) and introduces a language for representing EA components that lends itself to modelling *as-is* and *to-be* EA with a concrete aim to providing a simulation environment that delivers an unambiguous description of the required changes. The LEAP method and the language are illustrated with a detailed case study of business change currently being addressed by UK higher education institutions.

Enterprise Architecture (EA) aims to capture the essentials of a business, its IT and its evolution, and to support analysis of this information: *'it is a coherent whole of principles, methods, and models that are used in the design and realization of an enterprise's organizational structure, business processes, information systems and infrastructure'* [11].

A key objective of EA is being able to provide a holistic understanding of all aspects of a business, connecting the business drivers and the surrounding business environment, through the business processes, organizational units, roles and responsibilities, to the underlying IT systems that the business relies on.

In addition to presenting a coherent explanation of the *what*, *why* and *how* of a business, EA aims to support specific types of business analysis including: alignment between business functions and IT systems; business change describing the current state of a business (*as-is*) and a desired state of a business (*to-be*); maintenance the de installation and disposal, upgrading, procurement and integration of systems including the prioritization of maintenance needs; quality by managing and determining the quality attributes for aspects of the business such

as security, performance to ensure a certain level of quality to meet the needs of the business; acquisition and mergers describing the alignment of businesses and the aspects on both when they merge; compliance in terms of a regulatory framework, e.g. Sarbanes-Oxley; strategic planning including corporate strategy planning, business process optimisation, business continuity planning, IT management [8,16,13,2,10].

EA has its origins in Zachman's original EA framework [21] while other leading examples include the Open Group Architecture Framework (TOGAF) [18] and the framework promulgated by the Department of Defence (DoDAF) [20]. In addition to frameworks that describe the nature of models required for EA, modelling languages specifically designed for EA have also emerged.

A number of specialized modelling notations have been proposed for EA modelling. In most cases these notations provide a number of views or layers that capture the enterprise from different perspectives. The notations provide domain specific modelling languages (DSMLs) for EA and as such conceptualize a system in terms of domain elements and their

relationships. A representative example of such a DSML is ArchiMate [9].

Accompanying methods for EA have emerged but are not widely adopted, perhaps due to their size and complexity. Riege et al note the evolving nature of method requirements and the need to tailor methods for specific scenarios [16]. Methods, where they have been used, tend to be available as overarching large frameworks (similar to TOGAF) often located within consulting divisions of large corporations. Examples of such methods include: the Oracle Enterprise Architecture Framework¹ and IBM's EA Consulting Method². Both while attempting to be lightweight present large all-encompassing approaches to EA.

One area which existing method frameworks are not able to easily examine and address in a lightweight manner is the need to understand how to change an EA to meet a new requirement. Drilling down, the potential impact and change required would need to be promulgated as an impact analysis, a sliced view of the EA (of the systems affected), a gap analysis of missing functions and most importantly an equivalence analysis of an existing system and proposed changes. Method frameworks that have largely presented layered architectural models do not necessarily lend themselves to this type of modelling and analysis. Furthermore their bloated and document driven nature presents additional issues of complexity and places significant workloads on enterprise architects and those tasked with managing systems in large organization. The requirements for a lightweight method were discussed in detail in an earlier paper [5].

The architectural style of an organization has the potential to influence the construction and use of an EA. Several different styles of architecture are possible. A *Service Oriented Architecture* (SOA) involves the publication of logically coherent groups of business functionality as interfaces that can be used by components using

synchronous or asynchronous messaging. An alternative style, argued as reducing coupling between components and thereby increasing the scope for component reuse, is *Event Driven Architecture* (EDA) whereby components are event generators and consumers.

An important difference between SOA and EDA is that the latter generally provides scope for *Complex Event Processing* (CEP) where the business processes within a component are triggered by multiple, possibly temporally related, events. In SOA there is no notion of relating the invocation of a single business process to a condition holding between the data passed to a collection of calls on one of the component's interfaces. As described in [12] and [17], complex events can be the basis for a style of EA design. EDA replaces interfaces with events that trigger organizational activities. This creates the flexibility necessary to adapt to changing circumstances and makes it possible to generate new processes by a sequence of events [15].

Whilst a complex event based approach to architectural design must take efficiency concerns into account, the primary concern is how to capture, represent and analyze architectural information as an enterprise design. EDA and SOA are closely related since events are one way of viewing the communications between system components. The relationship between event driven SOA and EA is described in [1] where a framework is proposed that allows enterprise architects to formulate and analyze research questions including 'how to model and plan EA-evolution to SOA-style in a holistic way' and 'how to model the enterprise on a formal basis so that further research for automation can be done.' Our claim is that system architectures should be based on both EDA and SOA. We have described in detail the language for integrating EDA and SOA [6]. In that paper we present a unified language for modeling architectures, a corresponding simulation language and an environment for exploring simulations. We use that technology supported a light-weight method in this paper.

¹ <http://tinyurl.com/bpzo2u2>

² <http://tinyurl.com/cr4ph5m>

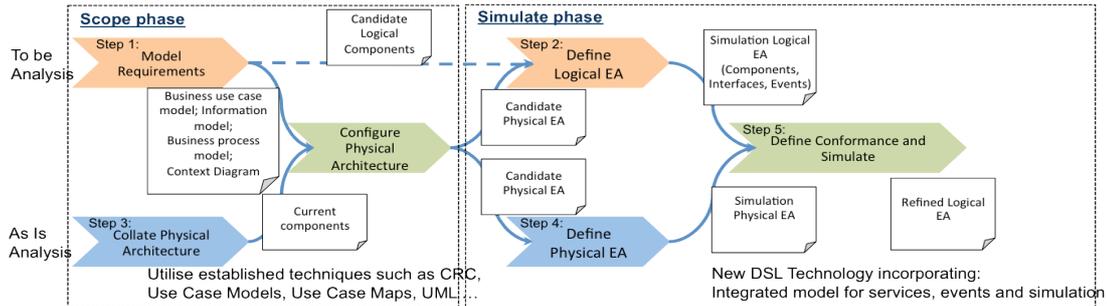


Figure 1 LEAP Method

Our contribution in this paper is to provide an agile method for EA that utilizes an integrated set of concepts derived from SOA and complex event processing that could easily be used as a standalone EA method or be integrated into larger EA frameworks. The underpinning concepts are well-formed with precise semantics that supports simulation of EA requirements. Of particular note, is that one of the key requirements facing enterprise architects, namely how to understand the mapping and therefore the solution space from an As-Is to a To-Be architecture and how it is achieved through equivalence modeling. Our method is validated through the use of a case study that is based on genuine requirements facing IT directors in all UK higher education institutions in 2012.

Our intention is that readers understand the method and its application easily so we have chosen to integrate the method and illustrations of its use by drawing upon the case study. Thus the paper is structured as follows. The section named *LEAP for Event Driven EA* introduces the proposed method for constructing an EA requirements (To-Be) and a physical EA (As-Is) and to compare the two using simulation. The section named *Case Study* presents the case study. The section named *Applying LEAP* then follows the method by applying it to the case study.

LEAP FOR EVENT DRIVEN EA

In this section we introduce our Lightweight EA method (LEAP) and the key technology concepts required for supporting agile EA development. The motivation for developing a method to support EA is driven from our knowledge and experience of the prevailing EA methods, namely that existing methods are large, cumbersome, lacking agility and not based on

well-defined concepts. Where existing methods have used modelling languages such as Archimate they are constrained by orthodox layering approaches (business layer, functional layer, deployment layer and so on). The layers prevent rigorous equivalence modelling in order to identify changes and new additions to an EA. Because of its simplicity, our proposed method may utilize existing techniques and approaches as appropriate.

Figure 1 shows an overview of our proposed method. Consistent with most approaches to EA methods where there is need to describe As Is and To Be models, there are two streams of activity, which converge at key stages. The *to-be* analysis stream includes activities to **Model Requirements**. We do not prescribe how you might wish to derive the requirements in order to produce a model of requirements but as our method is based on UML, models will include artefacts such as business information models, process models and business use case models. Existing method approaches such as Catalysis [7] and its derivatives [4] could be used for developing information models whilst recommended approaches for process modelling could include Ould's approach [14].

In parallel to the **Model Requirements** step, the activities in the **Collate Physical Architecture** step will bring together existing descriptions of systems and their configurations. Our experience of such descriptions is largely pictorial based documentation captured using drawing tools such as PowerPoint. A key output of this stage is a description of the systems that exist in the organization. We recommend capturing the description of each system as a UML Component to aid the migration to later stages of the method. Again, the method does not

prescribe modelling approaches; it leaves it to the practitioner to determine how to produce the required artefacts.

The **Configure Physical Architecture** step is an agile and simple way of slicing a description of an EA to determine what system components are likely to be impacted by emerging requirements. Techniques that can be used to support this impact analysis include use case maps [3]. A *use case map* is simply a trace of path of causal sequences of events across a set of system components representing an EA. The events are triggered by a business use case identified in the Model Requirements step. Alternative approaches that could be used in this step could be the use of CRC to help identify those system components that are (collaboratively) responsible for delivering a business use case [19]. The key output from this activity is an artefact expressed in system components that includes all the EA system elements that will be subject to some impact as a result of the emerging requirements.

Up to now, the steps in the method have utilized well-established notations and techniques. The subsequent steps in stage 2 incorporate an integrated set of concepts from SOA and complex event processing.

The **Define Logical Enterprise Architecture (L-EA)** step produces a model based description of a target logical EA - that is - the system components that are likely to be required as a result of the Model Requirements step. However, in recognition of the agile nature of the method, the Logical EA uses as a short cut, a candidate logical components from the Configure Physical Architecture step.

The Logical EA (L-EA) uses our integrated concepts derived from SOA and complex event modelling so the L-EA is expressed as components offering services, raised events, requested services and listened to events. Dependencies between components are thus expressed in terms of services request and fulfilments and event management. There may

also be constraints that will need to be fulfilled. These are expressed as invariants.

The **Conformance** step uses simulation to produce and visualize results. The logical architecture describes what is required and the physical to-be architecture defines how existing systems can be used to satisfy the requirements. It remains to validate the physical architecture by showing that the behaviour conforms to the requirements. If the simulation produces the same output when it is run with both the logical and physical definitions for university on a sufficiently large representative sample of data sets then we have confidence that the physical conforms to the logical. Such an approach presents a practical solution that is geared towards EA practitioners. Our EA design method produces a context and both a logical and a physical architecture description using the LEAP language.

CASE STUDY

Having outlined the method and concepts, this section presents a genuine requirement faced by IT directors in UK higher education institutions to deliver key information sets (KIS) to applicants deciding on which course and which university to chose for study at under graduate level. The case study is used throughout the rest of this paper as a basis for our proposed EA method.

Higher education institutions (HEI) in the UK are faced with a challenging and dynamic business environment where public funding of HEIs has been reduced by up to 70%. This lost funding is being replaced by the introduction of a new student fees regime beginning in 2012 following a bill introduced in the UK parliament in November 2010. The UK Government is of the view that students will require key information set (KIS) in order to make informed decisions regarding the selection of courses and institutions. Currently this information is not readily available in a consistent and easily accessible form. Consequently the Higher Education funding body (HEFCE) is

operations (E and H) for registration that cause a registration event to be raised (G and C). The registration events are processed by HEFCE *in any order*; both must be received before the KIS census date.

The KIS census date causes a timing event (I) that is processed by HEFCE causing it to request data necessary to build a KIS report. The NSS, University and DHLE components provide suitable operations (D, F and J) for requesting data.

questions such as: *'Do you agree that you receive prompt feedback on formative assessments?'*. Each course is delivered in terms of scheduled, guided and practical teaching and learning components, and assessed in terms of exams, courseworks and practicals. Information is maintained nationally about employment statistics for particular courses, such as the salary of graduates and the percentage who are in work or unemployed 6 months after graduation. Each HE course in the UK has a cost and may involve various forms of financial support.

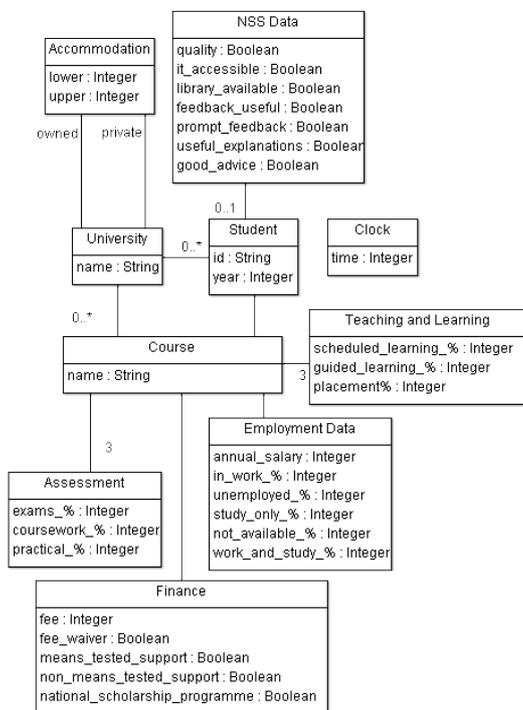


Figure 3 Information Model

Figure 3 shows the information model for the logical architecture. Each University has a number of students with unique identifiers and in a particular year of study. Information is maintained on the cost of both University owned accommodation and private accommodation in the area. A student studies a course and optionally completes an NSS return in their third year of study; the NSS form allows students to comment on the quality of the University's provision of teaching and learning in terms of

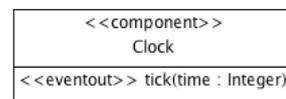


Figure 4 Clock

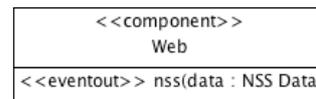


Figure 5 Online NSS Portal

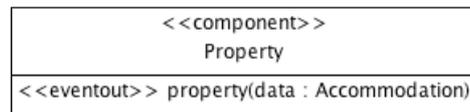


Figure 6 Property RSS Feed

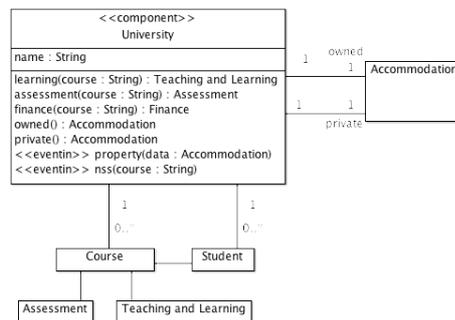


Figure 7 University

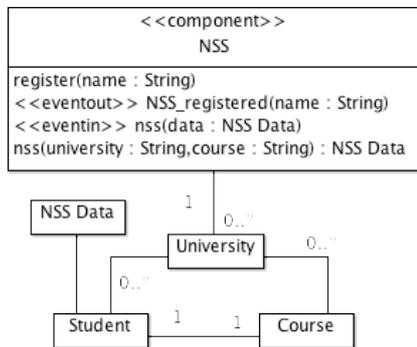


Figure 8 NSS

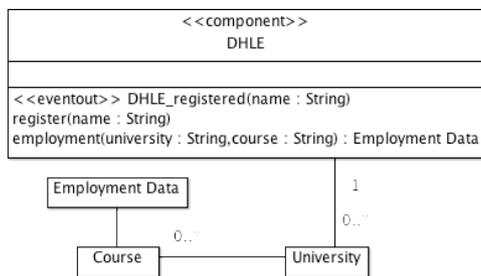


Figure 9 DHLE

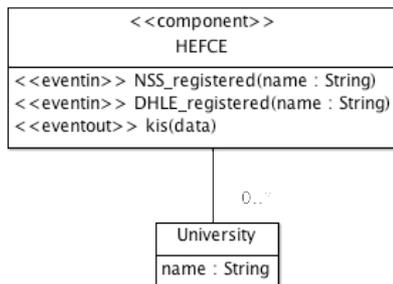


Figure 10 HEFCE

The LEAP simulation language uses components, records, operations and events to model an EA. Figure 4 to Figure 10 uses these concepts to elaborate the components from the context diagram in Figure 2 using the information concepts from Figure 3. Each diagram shows an individual component (identified using the stereotype <<component>>), its associated data model, the events it expects (<<eventin>>), the events it produces (<<eventout>>) and the operations that it supports.

Note that several information concepts occur in more than one component. In all cases, information is private to a component, and a system invariant is required if the information is to be consistent. For the purposes of our case study, the university maintains full information on courses and students and all references to these concepts are unique.

Step 2: Define L-EA

A key feature of our approach is to simulate both a logical and a physical architecture in terms of components, operations and events. To achieve this we have implemented a simulation language as a Java interpreter. A simulation model consists of a collection of component definitions of the following form:

```

component name (monitored components) {
  state { ...terms... }
  invariants { ...conditions... }
  operations { ...function defs... }
  rules { ...rule definitions... }
  ...nested definitions...
}
  
```

Each component monitors the events raised by a list of named components. A component maintains a private state that is a list of terms (named records). The component defines an initial state and may add and delete terms. The invariant conditions are boolean expressions over the component's state; a message is displayed when an invariant becomes false.

The operations of a component implement the business processes that form the component's interfaces; components invoke each other's operations by sending messages. A component monitors its state and the events it receives from components it monitors, using rules. Each rule has a collection of patterns that match the terms and events in the component's state. When all patterns are matched then the rule is ready to fire. The body of a rule is an action that can involve modifying the state or sending messages. A simple example example is:

```

component share_dealer (share_prices) {
  state {
    Shares(0)
    Money(1000)
    Buy_At(50)
    Sell_At(100)
  }
  invariants {
    never_broke {
      find Money(m) in state { m >= 0 }
    }
  }
  operations {
    get_shares() {
      find Shares(n) in state { n }
    }
  }
  rules {
    buy:
      Shares(n)
      Money(m)
      Price(p,avail)
      Buy_At(q)
      ?(p<q) ?(avail*p < m) {
        delete Shares(n);
        delete Money(m);
        new Shares(n + avail);
        new Money(m - (avail*p))
      }
    sell:
      Shares(n)
      Money(m)
      Price(p,_)
      Sell_
      At(q) ?(q < p) {
        delete Shares(n);
        delete Money(m);
        new Shares(0);
        new Money(m + (n*p))
      }
  }
}

```

To save space in the rest of the paper we will not give the complete definition of each component. Instead, we will provide those elements of the **state**, **invariants**, **operations** and **rules** sections that are essential to understanding how the simulation works.

The initial state of a clock is the term `Time(0)`. The clock provides a single operation that is used to drive the simulation. The operation provides an example of a new command that adds a new term to the state of the component. A clock has a single rule that fires when the clock

ticks. It increments the time and raises an event `Time(n)`, note the use of **delete** and **raise** to remove a term from the state of a component and to raise an event:

```

tick: t1=Tick() t2=Time(n) {
  delete t1,t2;
  raise Time(n);
  new Time(n+1)
}

```

A student uses the NSS web portal to complete the NSS survey. For simulation purposes the initial state of the web component contains terms defining the times at which students complete the survey, and the answers to the questions. The web component monitors the clock events and has a single rule that raises two events when a student completes a survey. The first event is processed by the NSS agency and the second is processed by the university:

```

step: Time(t)
  NSS(t,id,cid,qual,it,lib,
    help,prompt,expl,advice) {
    raise NSS(id,cid,qual,it,lib,
      help,prompt,expl,advice);
    raise NSS_complete(id)
  }

```

The property component simulates an RSS feed that supplies bounds on annual rental costs. The initial state is a collection of terms describing the times at which the data becomes available. The property component monitors clock events and raises an event when the appropriate time is reached using the following rule:

```

tick: Time(n)
  Property_Feed(n,low,high) {
    raise Property(low,high)
  }

```

The DHLE agency is responsible for maintaining employment data on universities and courses. For the purposes of the simulation all

```

register(university) {
  new University(university);
  raise DHLE_Registered(university)
}

```

employment data is part of the initial state. The DHLE component implements an operation that allows a university to register. It raises an event when this occurs.

HEFCE will request employment data when it constructs a KIS report. The operation provides an example of the **find** construct that is used to select an element from a list (in this case the state of DHLE) that matches a pattern; the body of the **find** expression returns a new term:

```
employment(uni,course) {
  find Employment(uni,course,s,w,
    u,so,na,was) in state {
    Employment(s,w,u,so,na,was)
  }
}
```

The NSS component provides an operation for registering a university:

```
register(university) {
  new University(university);
  raise NSS_Registered(university)
}
```

The HEFCE component uses the following operations to request KIS data from the NSS component. In each case a list-comprehension is used, for example:

```
[id | NSS(id,course,true,_,_,_,_,_,_)]
  <- state]

quality_score(course) {
  length([id |
    NSS(id,course,true,_,_,_,_,_,_)
  <- state])
}

it_score(course) {
  length([id |
    NSS(id,course,_,true,_,_,_,_,_)
  <- state])
}
```

In order for a university to be registered with HEFCE, it must have registered with both the NSS and DHLE components in any order as defined by this rule:

```
university: NSS_Registered(name)
  DHLE_Registered(name) {
  new University(name)
}
```

When the census date is reached the HEFCE component will make calls on NSS, University and DHLE components to get the KIS data:

```
kis: Time(t)
  KIS_census(t) {
  get_kis_data()
}
```

The component definitions above provide the context for the logical definition of a university given in this section. The initial state of a university is shown below. It defines several distinct types of information: the times at which the university registers with the government agencies; the financial, teaching and learning, and assessment patterns for each course; the cost of university owned accommodation; the cost of private accommodation; student records that assign a unique id to each student, assign them to a course, define their year of study and record whether or not they have completed the NSS survey:

```
Register(nss,2) // Register NSS at 2.
Register(dhle,3) // Register DHLE at 3.
Course('p01',8500,true,false,true,false)
Learning('p01',[60,30,35],
  [20,20,50],
  [20,50,15])
Assessment('p01',[70,50,50],
  [10,20,30],
  [20,30,20])
...
Lower_Accommodation(7000)
Upper_Accommodation(9600)
Lower_Private(0)
Upper_Private(0)
Student('s01','p01',3,false)
Student('s02','p01',3,false)
...
NSS_census(5)
```

A university includes a number of invariant conditions that must be met at all times. The following is an example called `assessments` that requires the split between %-age values for examinations, courseworks and practicals in any given year to add up to 100% for each course. The condition uses the `forall` expression that matches a pattern against all elements in a list; the body of the `forall` must return `true` for each element in the list that matches the pattern `trans` is defined in the operations section below:

```
assessments {
  forall
    Assessment(course,exams,coursewks,pracs)
  in state {
    forall [e_pc,c_pc,p_pc] in
      trans ([exams,coursewks,pracs]) {
        (e_pc + c_pc + p_pc) = 100
      }
  }
} else 'all assessments must add up to 100
in each year.'
```

A second invariant encodes a business requirement that 50% of all students registered for a course must complete the NSS survey in year 3:

```
nss_50_per_course {
  forall Course(id,_,_,_,_) in state {
    find UniTime(t) in state {
      find NSS_census(t) in state {
        let nss = length([id |
          Student(_,id,3,true) <- state]);
          students = length([id |
            Student(_,id,3,_) <- state])
        in if students = 0
          then true
          else ((nss/students)*100) > 50
        } else true
      } else true
    }
  } else ' 50% of all students must complete
the NSS.'
```

```
trans(lists) {
  if exists l in lists { l = [] }
  then []
  else [ head(l) | l <- lists ] :
    slice([ tail(l) | l <- lists ])
```

The following operations are used by HEFCE to access the data necessary in a university to produce the KIS report:

A university has the following rules: `register`

```
learning(course) {
  case find Learning(course,_,_,_) in state {
    Learning(n,sched,guided,plmnt) ->
    Learning(sched,guided,plmnt)
  }
}
assessment(course) {
  case find Assessment(course,_,_,_) in state {
    Assessment(n,exams,cw,pract) ->
    Assessment(exams,cw,pract)
  }
}
finance(course) {
  case find Course(name,_,_,_,_,_) in state {
    Course(n,f,fw,mt,nmt,sch) ->
    Finance(f,fw,mt,nmt,sch)
  }
}
```

that registers with the appropriate government agencies when the appropriate time event occurs; `clock` that is used to keep a local record of time; `property` that is used to update the prices of private accommodation using events from the RSS feed; `nss` that is used to record the completion of an NSS survey by a particular student:

```
register: Register(target,time)
  Time(time) {
    send(target,'register',[self.name()])
  }
clock: Time(time) {
  delete UniTime(time-1);
  new UniTime(time)
}
property: p=Property(low,high)
  l=Lower_Private(x)
  u=Upper_Private(y) {
  delete p,l,u;
  new Lower_Private(low);
  new Upper_Private(high)
}
nss: NSS_completed(id)
  s=Student(id,course,3,false) {
  delete s;
  new Student(id,course,3,true)
}
```

SIMULATION

Our approach uses simulation to show that a physical architecture is consistent with the behaviour required by a logical architecture.

Control

Select a course
 p03
 Advance simulation
 Step:7.p03

GRADUATE EMPLOYMENT

£20000 Average annual salary after 6 months
 90% go on to work and/or study

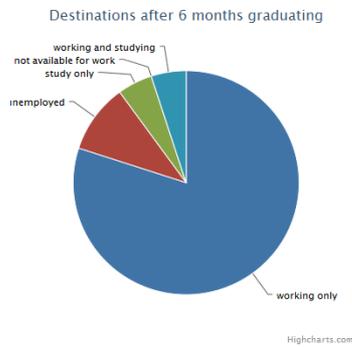
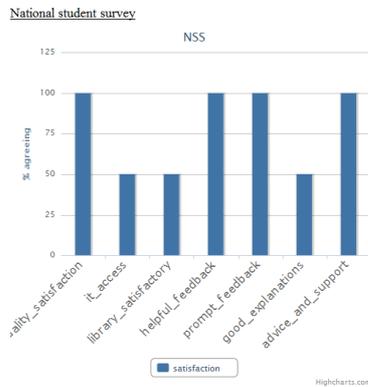


Figure 11 Simulation (Part 1 of 3)

STUDENT SATISFACTION



FINANCIAL

£ 7500 Tuition fees, per year
 Financial support available
 Fee Waiver true
 Means Tested Support false
 Non Means Tested Support false
 National Scholarship Programme false
 £ 7000 - £ 9600 Average annual cost of institution accommodation
 £ 6400 - £ 8750 Private rental market

Figure 12 Simulation (Part 2 of 3)

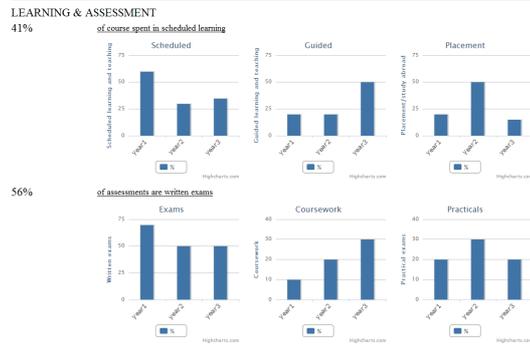


Figure 13 Simulation (Part 3 of 3)

The LEAP language for the KIS context and logical architecture has been defined in previous sections. LEAP also provides features for generating a GUI that can be deployed on a web server and accessed via a web browser. We do not show the LEAP code for the KIS GUI, but the output is shown in Figure 11 to Figure 13. The control section in Figure 11 is used to step through the simulation by sending tick() messages to the clock. The step button shows the current time and the currently selected course; the rest of the GUI shows KIS information for the course.

Step 3: Collate Physical-EA

The next step of our method involves reviewing the current physical as-is system architecture. Most organizations have a systems overview that is used as the input to this step. The result is an understanding of the currently capability of the organization in terms of systems, interfaces, information and events.

We will use the University of Middlesex (Mdx), London, UK as the basis for our case study. Space limitations prevent us from providing a complete description of the Mdx physical architecture, however it is consistent with most UK HEIs and includes systems for registry, an asset management system that includes a sub system for university accommodation, an examinations database, a library system, a financial management system called PAFIS, a teaching and learning system called OASIS, an

alumni management system, a student portal, and a staff portal.

Step 4: Configure Physical-EA

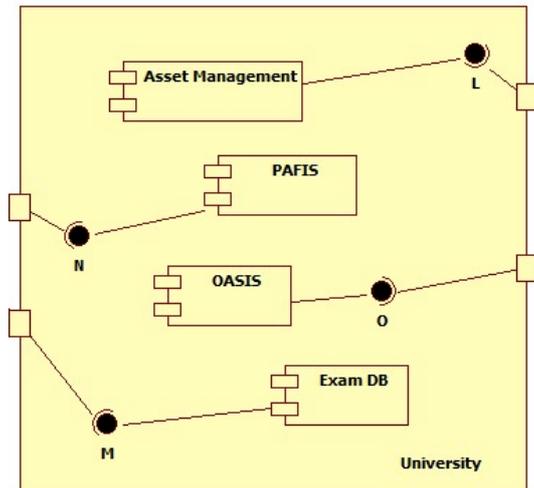


Figure 14 Physical Architecture

The next step of the method analyses the physical system of an organization and takes an appropriate *slice* to produce just those systems that will be involved in the required-to-be architecture. In the case of supplying KIS data, we know that Mdx will need to provide student, accommodation, teaching and learning, assessment and financial information. Therefore, for example the P-EA will not include the alumni or library management systems.

The P-EA for Mdx is shown in Figure 14 where interfaces have been introduced that support the appropriate delegation of the operations defined by the university component. The next section defines the components using LEAP.

Step 5: Define Physical-EA

The previous section has identified the slice of the Mdx systems that are required to support the KIS case study. This section defines each component using LEAP.

The asset management system must provide an interface for accommodation pricing. The part of the assessment information that was

```

component asset_management() {
  state {
    Lower_Accommodation(7000)
    Upper_Accommodation(9600)
  }
  operations {
    lower_accommodation() {
      find Lower_Accommodation(cost)
      in state { cost }
    }
    ...
  }
}

```

handled by the university in the logical model is delegated to the exams database:

The Mdx financial management system PAFIS is

```

component exam_db() {
  state {
    Exam('p01', [70,50,50])
    ...
  }
  operations {
    find_exams(name) {
      find Exam(name,exams) in state
    }
  }
}

```

used to handle information about course costs and support:

The Mdx teaching and learning system OASIS is

```

component pafis() {
  state {
    Course('p01',8500,true,false,true,false)
    ...
  }
  operations {
    finance(course) {
      case find Course(course,_,_,_,_,_)
      in state {
        Course(n, f, fw, mt, nmt, sch) ->
          Finance(f, fw, mt, nmt, sch)
      }
    }
  }
}

```

used to handle the teaching and assessment patterns:

The definition of University is then modified to

```

component oasis() {
  state {
    Learning('p01',
      [60,30,35],
      [20,20,50],
      [20,50,15])
    ...
    Assessment('p01',
      [10,20,30],
      [20,30,20])
    ...
  }
  operations {
    find_learning(name) {
      find Learning(name,_,_) in state
    }
    find_assessment(name) {
      find Assessment(name,_,_) in state
    }
  }
}

```

include the components defined above as sub-definitions. The resulting component is the physical architecture description of University and must include some new invariants. Firstly, information about courses is now distributed amongst a number of systems. We must ensure that the information is consistent, therefore we introduce a new university invariant conformance (seteq14 is a predicate that holds between four lists when they all have the same elements):

```

conformance {
  seteq14(
    [id | Exam(id,_) <- exam_db],
    [id | Assessment(id,_,_) <- oasis],
    [id | Learning(id,_,_,_) <- oasis],
    [id | Course(id,_,_,_,_) <- pafis])
  } else 'inconsistent information.'
}

```

Finally, information about assessment for a given course must add up to 100% in any year. The information is distributed between the OASIS and exams systems:

```

assessments {
  forall Assessment(course,coursewks,pracs)
  in oasis {
    forall Exam(course,exams) in exam_db {
      forall [e_pc,c_pc,p_pc]
      in slice([exams,coursewks,pracs]) {
        (e_pc + c_pc + p_pc) = 100
      }
    }
  }
} else 'all assessments must add up to 100'

```

Step 6: Conformance

Our EA design method produces a context and both a logical and a physical architecture description using the LEAP language. The logical architecture describes what is required and the physical to-be architecture defines how existing systems can be used to satisfy the requirements. It remains to validate the physical architecture by showing that the behaviour conforms to the requirements.

In general, conformance can be established using a number of approaches. The context defines a collection of system executions in terms of messages, events and state changes. It is possible to use inspection-based techniques to show that all required executions are handled appropriately by the physical architecture.

More precise approaches are possible. For example, we can define mappings from the physical architecture components to the logical architecture. In the case of the Mdx architecture, the `asset_management`, `exam_db`, `pafis` and `oasis` components are merged to become a single top-level university component. Providing that all operations, events and invariants are consistent then the physical architecture is conformant.

Another approach is to rely on simulation. LEAP can be used to run the simulation to produce GUI output as defined in the simulation figures above. If the simulation produces the same output when it is run with both the logical and physical definitions for university on a sufficiently large representative sample of data sets then we have confidence that the physical conforms to the logical. When we run the simulation with both definitions of university given in this paper, the output is the same.

Finally, if we require total confidence in conformance then we need to resort to formal methods such as model checking and theory proving. For large systems such as those found

in EA, formal methods are often impractical in terms of complexity. That said, a formal semantics for LEAP is an area for future development in order to investigate whether formal methods could help.

CONCLUSION

Enterprise Architecture remains a confusing and constantly evolving collection of methods and frameworks that are generally characterized by an expansive outlook, lack of precision, a focus on diagrams and an emphasis on document management. The result is that existing approaches are difficult to use. This paper has presented an effort to reduce the scope of EA in order to pin down the core use cases of managing change and better understanding the impact of changing requirements on existing technical architectures of an organization. We have proposed a lightweight EA method and its accompanying language that supports precise description of *to be* and *as is* EA. The language by virtue of its design can be used with a simple Java based simulation environment to experiment with conformance of proposed EA changes. We recognise that there are several limitations with our current proposal but we have demonstrated using a detailed case study that there is merit and considerable potential in using such an approach understanding and managing EA.

BIBLIOGRAPHY

- [1] M. Assmann and G. Engels. Transition to service-oriented enterprise architecture. *Software Architecture*, pages 346–349, 2008.
- [2] T. Bucher, R. Fischer, S. Kurp juweit, and R. Winter. Analysis and application scenarios of enterprise architecture: An exploratory study. In *10th IEEE International Enterprise Distributed Object Computing Conference Workshops*, 2006. EDOCW'06, 2006.
- [3] R.J.A. Buhr and R.S.O. Casselman. Use case maps for object-oriented systems, volume 302. Prentice Hall, 1996.
- [4] J. Cheesman and J. Daniels. *UML components*. Addison-Wesley, 2001.
- [5] T. Clark, B.S. Barn, and S. Oussena. Leap: a precise lightweight framework for enterprise architecture. In *Proceedings of the 4th India Software Engineering Conference*, pages 85–94. ACM, 2011.
- [6] Tony Clark and Balbir S. Barn. A common basis for modelling service-oriented and event-driven architectures. In *5th India Software Engineering Conference*, IIT Kanpur, ISEC 12, 2012.
- [7] Desmond F. D'Souza and Alan Cameron Wills. *Objects, components, and frameworks with UML: the catalysis approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [8] M. Ekstedt, P. Johnson, A. Lindstrom, M. Gammelgard, E. Johansson, L. Plazaola, E. Silva, and J. Lilieskold. Consistent enterprise software system architecture for the cio - a utility-cost based approach. In *System Sciences*, 2004. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, 2004.
- [9] The Open Group. Archimate technical standard, <http://www.opengroup.org/archimate/>. 2008.
- [10] J.C. Henderson and N. Venkatraman. Strategic alignment: Leveraging information technology for transforming organizations. *IBM systems Journal*, 32(1), 1993.
- [11] Marc Lankhorst. Introduction to enterprise architecture. In *Enterprise Architecture at Work, The Enterprise Engineering Series*. Springer Berlin Heidelberg, 2009.
- [12] B.M. Michelson. Event-driven architecture overview. Patricia Seybold Group, 2006.
- [13] K.D. Niemann. From enterprise architecture

to IT governance: elements of effective IT management. Vieweg+ Teubner Verlag, 2006.

[14] M.A. Ould. Business Process Management: a rigorous approach. British Informatics Society Ltd, 2005.

[15] S. Overbeek, B. Klievink, and M. Janssen. A flexible, event-driven, service-oriented architecture for orchestrating service delivery. IEEE Intelligent Systems, 24(5):31–41, 2009.

[16] C. Riege and S. Aier. A Contingency Approach to Enterprise Architecture Method Engineering. In Service-Oriented Computing–ICSOC 2008 Workshops. Springer, 2009.

[17] G. Sharon and O. Etzion. Event-processing network model and implementation. IBM Systems Journal, 47(2):321–334, 2008.

[18] J. Spencer et al. TOGAF Enterprise Edition Version 8.1. 2004.

[19] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. Designing object-oriented software, volume 13. Prentice Hall Englewood Cliffs, New Jersey, 1990.

[20] DE Wisnosky and J. Vogel. DoDAF Wisdom: A Practical Guide to Planning, Managing and Executing Projects to Build Enterprise Architectures Using the Department of Defense Architecture Framework (DoDAF), 2004.

[21] J.A. Zachman. A framework for information systems architecture. IBM systems journal, 38(2/3), 1999.