LEAP: A Precise Lightweight Framework for Enterprise Architecture

Tony Clark
School of Engineering and Information Sciences
Middlesex University
Hendon, London, UK, NW44BT
t.n.clark@mdx.ac.uk

Balbir S. Barn
School of Engineering and Information Sciences
Middlesex University
Hendon, London, UK, NW44BT
b.barn@mdx.ac.uk

Samia Oussena
School of Computing
University of West London
St. Mary's Road, Ealing, London, UK
samia.oussena@tvu.ac.uk

ABSTRACT

This paper proposes LEAP: a simple framework for Enterprise Architecture (EA) that views an organization as an engine that executes in terms of hierarchically decomposed communicating components. The approach allows all aspects of the architecture to be precisely defined using standard modelling notations. Given that the approach is simple and precisely defined it can form the basis for a wide range of EA analysis techniques including simulation, compliance and consistency checking. The paper defines the LEAP framework and provides an overview in terms of a case study. LEAP does not mandate any specific notation, a UML-style notation is used in this paper and the implications for ArchiMate are analysed.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; C.0 [**Computer Systems Organization**]: Systems Architectures

General Terms

Modelling

Keywords

Enterprise Architecture, Model Driven Engineering

1. INTRODUCTION

Enterprise Architecture (EA) aims to provide a holistic view of a business from the business drivers through to their implementation using information systems. The leading technologies for EA provide diagram-based models for expressing architectures, however the diagrams are not based on a precise semantics and therefore can become difficult to manage when, inevitably, EA models become large and complex. Key features of EA technologies are that they provide different views of an organization and that they link the business motivators (goals) to their realization in technology. Model driven approaches can help to add precision to EA by using models to link various views and by defining semantics for the modelling languages.

This paper reviews EA in section **??** including a leading EA technology (ArchiMate in section **??**) and one of its proposed extensions (section **??**). The key problem we address is one of precision in EA technologies (section **??**). Our contribution is the definition in section **??** of a precise modelling language for EA, called LEAP, that can be mapped directly to leading EA technologies. LEAP is used in section **??** to develop and analyse part of an architecture for a university. The proposed extension to ArchiMate involves adding business goals and we show in section **??** how these can expressed and formally analysed in LEAP.

ISEC 2011 India

2. AN OVERVIEW OF EA

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 realisation of an enterprise's organizational structure, business processes, informaion systems and infrastructure.*' [**?**]

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 Alignment between business functions and IT systems; identification of inconsistencies and missing elements.

Business Change EA is often used to describe 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 Managing and determining the quality attributes for aspects of the business such as security, perfor-

mance to ensure a certain level of quality to meet the needs of the business.

Acquisition and Mergers Businesses often need to plan to acquire other businesses (or indeed plan to be taken over). EA can describe the alignment of businesses and the effect on both when they merge.

Compliance Businesses exist in a regulatory environment where external constraints are imposed on how the business operates. An example of regulations is the Sarbanes-Oxley Act of 2002 that assigns personal responsibility to senior management of public and non-public organizations in the U.S.

Strategic Planning Corporate strategy planning, business process optimisation, business continuity planning, IT management.

EA has its origins in Zachman's original EA framework [**?**] while other leading examples include the Open Group Architecture Framework (TOGAF) [**?**] and the framework promulgated by the Department of Defense (DoDAF) [**?**]. In addition to frameworks that describe the nature of models required for EA, modeling languages specifically designed for EA have also emerged. One leading architecture modelling language is Archimate [**?**].

2.1 Enterprise Architecture Modelling

The first step in EA analysis is to construct a model of a business. The model should describe features such as constraints, alternatives, impact or feasibility which will help in decision making processes. If the EA cannot support such rational decision-making then it cannot be seen as a *good* model. Thus there are some key requirements on the current technologies that are used for EA activity. These requirements can be summarised as the following and are consistent with that identified by Johnson et al. [**?**]:

- To support decision making, an EA language requires that the notation will support the decision-maker's goals, the domain in which the decision is required and the causal relations between that which is controlled and the decision.
- The language must display precision such that there is clarity on any of the represented concepts.
- The language must exhibit behavioral semantics - that is support how concepts affect each other as a result of actions.
- The language must support representation of concepts at different levels of abstraction.

As systems supporting business become increasingly more significant and complex an important approach to management and planning of systems that has gained prominence is model-based Enterprise Architecture (EA). An Enterprise Model is a computational representation of the structure, activities, processes, information, resources, people, behaviour, goals and constraints of a business, government, or other enterprise. It can be both descriptive and definitional - spanning what is and what should be. The role of an enterprise model is to achieve model-driven enterprise design, analysis and operation [**?**].

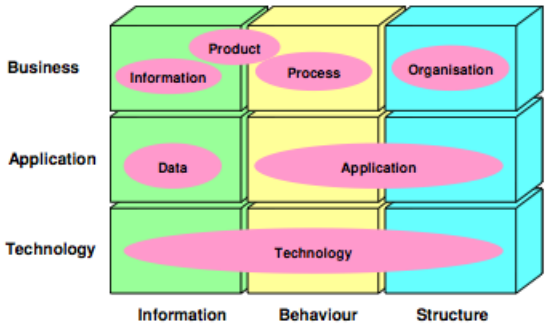

Figure 1: Basic ArchiMate Modelling

As described in [**?**]: '*Holistic approaches to EA deploy a multi-level framework or a hierarchy of design layers in order to represent the different views of an enterprise [...]. But these multi-layer approaches are often rather abstract, do not consider all necessary design layers, or do not specify consistency in adequate rigor.*'

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 provide a good conceptual fit to the problem of representing EA domain elements and their relationships. A representative example of such a DSML is ArchiMate [**?**] as described in the next section.

2.2 ArchiMate

ArchiMate [**?**] [**?**] is a standard managed by the Open Group (`http://www.opengroup.org/archimate`). It consists of a framework of layers and aspects similar to the Zachman framework [**?**] that defines a theory or 'world view' about the way enterprises are structured. The aspects are described using a set of modelling concepts that constitute a DSML for EA. The framework is described in figure **??** which is taken from [**?**]. The framework distinguishes between the business layer (concerned with products and services offered to customers), the application layer (concerned with the application services that the company implements internally), and the technology layer (concerned with the infrastructure services necessary to run the applications). To model each layer, Archimate provides model elements that express information, behaviour and structure. ArchiMate provides modelling concepts in each of the three layers:

Business actor; role; collaboration; interface; object; process; function; interaction; event; service; representation; meaning; value; product; contract.

Application component; collaboration; interface; object; function; interaction; service.

Technology node; device; network; communication path; interface; software; service; artifact.

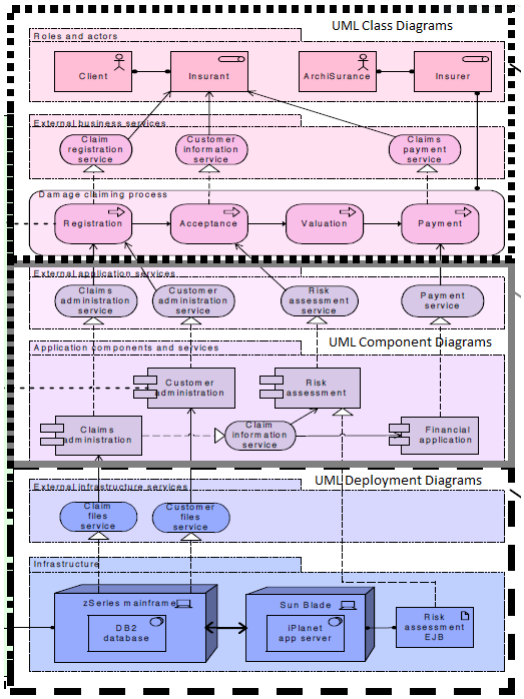

Figure 2: UML ArchiMate Profile

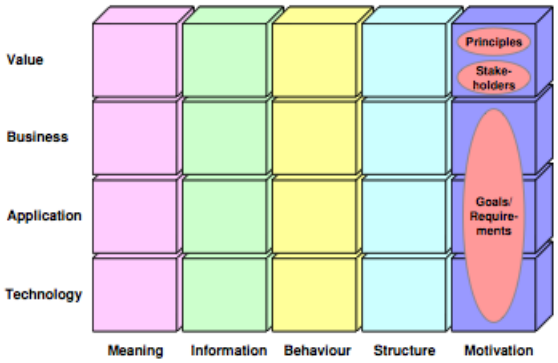

Figure 3: Extended ArchiMate Modelling

Figure 4: Goal Representation

Clearly there is a great deal of overlap between the modelling concepts in the different layers; for example, *interface* occurs in all three layers, and *function* occurs in the first two. There are concepts whose meaning would seem to overlap, for example *process*, *service* and *function*. Archimate provides a notation for each of the modelling concepts. The notation has a syntax definition in the form of well-formedness constraints, however there is no semantics in the sense of axiomatic, denotational or operational definitions.

There is a proposal for a UML profile for ArchiMate [**?**]. An overview of the proposal is shown in figure **??** where the business layer is modelled using class diagrams, the application layer using component diagrams and the technology layer using deployment diagrams. However this does not constitute a semantics since UML itself does not have a precise semantics. The lack of semantics makes it difficult to compare model elements for similarity, difference and redundancy. Futhermore, the lack of semantics makes it difficult to determine the meaning of extensions to the language as proposed in the next section.

2.3 Extending ArchiMate

Engelsmanb et al. [**?**] propose an extension to Archimate shown in figure **??** that is motivated by features from Goal Based Requirements Engineering [**?**] and the Business Motivation Modelling (BMM) notation [**?**] defined as a standard by the OMG. Goals are considered as high-level objectives of some organization or system. Goals are conditions to be satisfied by some aspect of a system or or a system modification. Goals can be decomposed into and-or trees and can be used with respect to different viewpoints of a system [**?**, **?**].

The extension to ArchiMate for business motivation proposed in [**?**] allows goals to be added to the business layer of an EA model. The proposed notation allows goals to be added to a model as boxes containing free format text. Goals may be decomposed as trees. The proposal claims to be able to test whether goals conflict. The example discussed in [**?**] involves hospital patient records where the goal of compliance to security guidelines conflicts with free access to patient data as shown in figure **??** can be shown to conflict.

As it stands, ArchiMate provides three aspects: *information*, *behaviour*, and *structure*. The proposed extension adds *motivation* and *value* as a new aspect and a new layer respectively. Motivation expresses features such as goals, objectives and directives in the different layers so that the reasons for decisions and the conditions under which business processes correctly achieve the business requirements can be clearly stated and tested. The extension is defined in [**?**] with respect to a meta-model and then used to express motivation in a hospital case study where goals are decomposed.

2.4 Problem and Contribution

ArchiMate represents a domain specific language for expressing EA, however it suffers from: (1) overlapping concepts; (2) lack of precision and semantics; (3) lack of refinement relationships between layers. The business motivation extension to ArchiMate described in section **??** incorporates mechanisms that have been used elsewhere in requirements engineering and system modelling, and the resulting extension allows conditions to be expressed that must be satisfied by any suitable EA model. However, unlike many system engineering approaches, Archimate is not sufficiently precise to allow the satisfaction criteria to be tested. Verification is left to human interpretation, which means that complex architectures can be prone to error. It is also difficult to see how tooling can be developed to support motivation modelling.

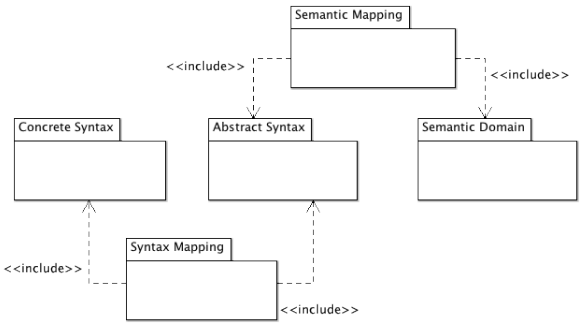

Figure 5: Precise Meta-Modelling

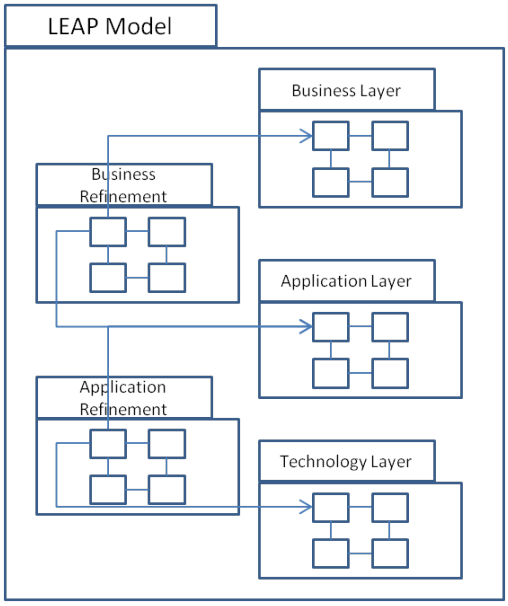

Figure 6: LEAP Models

Our contribution is to use a precise meta-modelling approach [**?**] to define a simple EA modelling language based on the Unified Modelling Language (UML), that addresses the problems described above. UML is a good fit for EA as shown in the ArchiMate profile for UML. The approach is outlined in figure **??** where a language is modelled in terms of concrete syntax, abstract syntax and semantics. The relationships between the component models of a language are defined as mappings.

The language is called LEAP and provides a minimal, but precisely defined collection of concepts that can be mapped to ArchiMate model elements. Business motivation that has been proposed as an extension to ArchiMate is defined in LEAP using the Object Constraint Language (OCL) standard that is part of UML. OCL is a formal language that can be used to precisely define the semantics of motivation and therefore to check consistency and verify EA models. This paper defines the LEAP language and shows how it can be used to express an EA model, including business motivation, using a case study.

3. LEAP: LIGHTWEIGHT, PRECISE EA

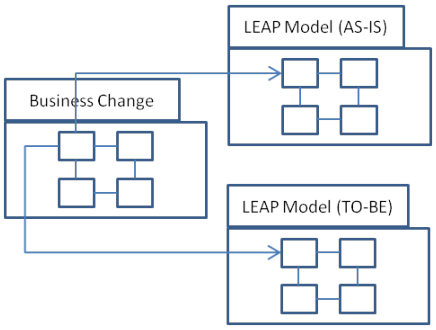

Figure 7: Business Change

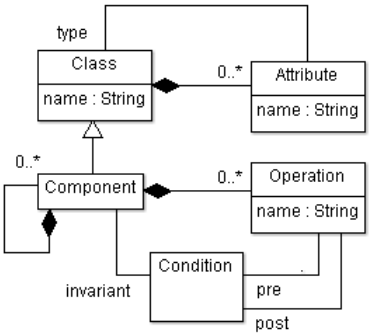

Figure 8: Layer Models

3.1 Overview

Fig **??** shows an overview of the models involved in using LEAP to express an enterprise architecture. LEAP is a model based approach and therefore all components of the approach are modelled starting with the business layer which captures the key information relating to the EA analysis being undertaken. The business layer captures the business concepts and the constraints relating to business drivers, directives and processes. Typically, a business layer will not include details of organizational structure.

The application layer is a *refinement* of the business layer. A refinement is a more detailed viewpoint of the same organization; typically, a business layer refinement will add organizational structure and associated business processes to the information contained in the business layer. It is important that the information in the application and business layers are consistent. To ensure this, the refinement is modelled. The business refinement model contains elements that link business and application concepts, ensuring that no information is lost.

The technology layer is a refinement of the application layer. Typically it introduces further detail by mapping the required logical business components to their physical realization in the form of IT systems. Like the business refinement, the application refinement is modeled and links elements to ensure that no information is lost.

LEAP can be used to analyse business change. Figure **??** shows two LEAP models. The first model is used to describe the current state of the organization and the second is used to describe a desired configuration of the organization. The relationship between the two is a business change model that

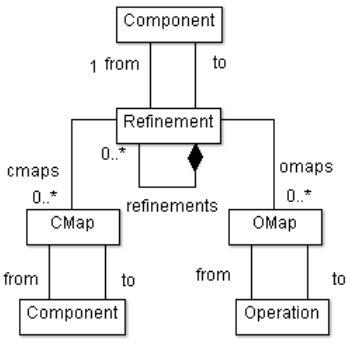

Figure 9: Refinement Between Layers

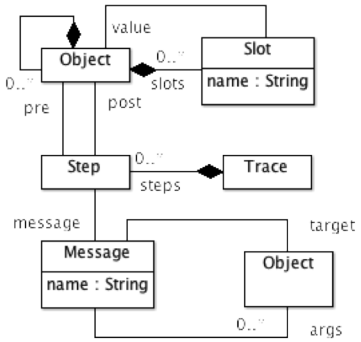

Figure 10: Layer Semantics

links elements in the *as-is* state to elements in the *to-be* state.

A feature of LEAP that differentiates it from other approaches to EA, for example the ArchiMate approach, is the requirement to model the relationships between the different layers and organization states. These models force an enterprise architect to be specific about the features of the organization that are being expressed in different viewpoints of a system. In the case of business change, the models clearly explain what changes are necessary.

In addition to modelling the relationships between views and states, LEAP models have a semantics that allows the models to be checked.

3.2 Modelling Layers

Leap takes some of its inspiration from Catalysis [**?**] including modelling a system as a component and system development as step-wise refienement. Each component represents a coherent grouping of information and behaviour in the organization. Constraints are used to specify the required behaviour and express invariants over the information held in the components. Refinements between layer models increase the level of detail; refinement is repeatedly applied until the components are mapped to physical systems (existing or yet to be implemented). In ArchiMate and related approaches, there are three layers; however; LEAP is not limited to three layers and can accommodate any number of refinements.

The meta-model for LEAP is shown in figure **??**. It provides the essential features of both class and component UML models. As discussed in [**?**] the first two layers of ArchiMate can be expressed using UML class and component models respectively. The third layer (technology) can be expressed using UML deployment models which are essentially component models. We therefore propose that a combination of class and component models can be used to express essential ArchiMate-style EA models extended to more than three layers where appropriate.

In LEAP layer models the entire organization is a top-level component. Components may contain both classes and components. Contained classes are used to define the information managed by a component. Sub-components are used to define logical and physical sub-systems in an organization. The behaviour of a component (business rules) are defined as operations in LEAP, where the behaviour of an operation is specified by a pre and post condition. A component has an invariant that places constraints on its structure and behaviour.

LEAP uses conditions in two ways: as *invariants* and as *pre* and *post conditions* on operations. In both cases the conditions are expressed using the Object Constraint Language (OCL) [**?**]. This allows constraints to captured in a formal language that can be checked precisely.

3.3 Refinement

LEAP layer models are related by refinement. A refinement maps different views of a system from a high-level view to a lower-level view. In ArchiMate, refinement is not precisely defined. In LEAP the relationships between the two layers must be modelled. The refinement meta-model is shown in figure **??** where a refinement relationship holds between a `from` component (the high-level layer) and a `to` component (the low-level layer). The refinement relationship has two elements: `cmaps` and `omaps` that relate components and operations respectively. A constraint requires that all the high-level elements are mapped by the refinement but that the lower-level component may contain more elements:

```
context Refinement inv:
  from.components = cmaps.from and
  from.components.operations = cmaps.operations and
  refinements.from = from.components
```

Although the refinement mapping can be expressed as a diagram, it is more convenient to express it using a textual syntax with the following format:

```
refine <layer>(<high-level>,<lower-level>) components :
 <cmap constraints>
refine <layer>(<high-level>,<lower-level>) operations :
  <omap constraints>
```

where the `<cmap constraints>` and `<omap constraints>` are OCL expressions denoting instances of `CMap` and `OMap` respectively.

3.4 Semantics

The semantics of each architecture layer is defined as a collection of traces. A trace is a sequence of steps. Each step describes the system state change that occurs when a message is sent to a component. A message consists of the target component, the name of the message (an operation defined by the target) and a sequence of operation argument values. The model used to express layer semantics is defined in figure **??**.

We use the semantics later in this paper to show how a refinement can be verified. We want to be as precise as

possible when describing the verification and therefore make use of the semantics model. One way to do this is to draw out object diagrams that are instances of the semantic model. However, this takes up a great deal of space. A more succinct way of expressing instances of models is using a text format where an instance has the form:

```
(C,o)[n=v;...] when Q
```

where `C` is the name of a class, `o` is an object identifier, `n` is the name of a field (or role end), `v` is a value (another object, atomic value or a sequence of values `[v,w,...]`), and `Q` is an OCL constraint. A key feature of this approach is that `o`, `v` and `Q` may contain free-variables allowing the instance to be a pattern, for example:

```
(Object,o)[
  slots=[
    (Slot,s1)[name='me';value=o],
    (Slot,s2)[name='age';value=x]]]
      when x < 100
```

This instance pattern corresponds to an object with two slots. The first slot is called `me` and references the object (note the self reference `o`) and the second slot is named `age` and has any value greater than `100`. Note that we may omit slots and object identifiers when they are not important.

LEAP is a precise modelling language in the sense that it supports constraints written in OCL, has a semantic domain, and requires relationships between different views to be modelled. The next section shows how LEAP is used to perform *business change analysis* in terms of *as-is* and *to-be* models.

4. CASE STUDY

The University of Middle England (UME) is worried about its future. After some analysis the key reason seems to be that students find the resources are used for teaching and learning activities to be badly aligned and that some of the modules assume that all students have their own lap-tops (which is not always the case).

As a result of this analysis UME decides to implement a lap-top loan scheme whereby students who do not have their own facilities can loan them from UME at no cost. The scheme raises questions about which modules require lap-tops, whether the current teaching and learning rooms are suitable for lap-top usage, and whether UME has suitable software with appropriate licenses for use by students.

In order to implement the lap-top scheme, UME decides to use EA. Having used EA to design a migration path, UME wants to determine whether the scheme is consistent with its business goals and to understand how its new architecture operates.

4.1 As-is Analysis

The first step in EA analysis is to get a clear understanding of the current state of the organization. A working group is tasked by the UME executive to come up with a description of the information structures relating to student teaching and learning. The result is shown in figure **??**.

There is a single component `university` that defines the essential information structures and operations that are likely to be effected by the introduction of the lap-top loan scheme. The model defines the information describing the as-is state in the *business-layer* as defined by ArchiMate. The goals of the layer are specified as an OCL invariant:

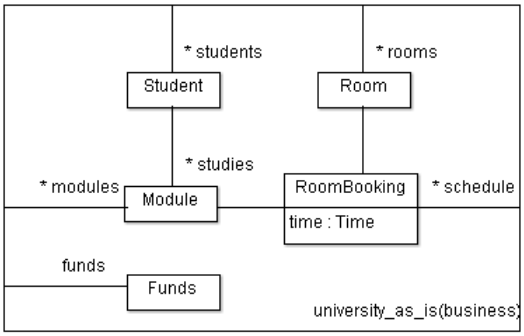

Figure 11: As Is (Business)

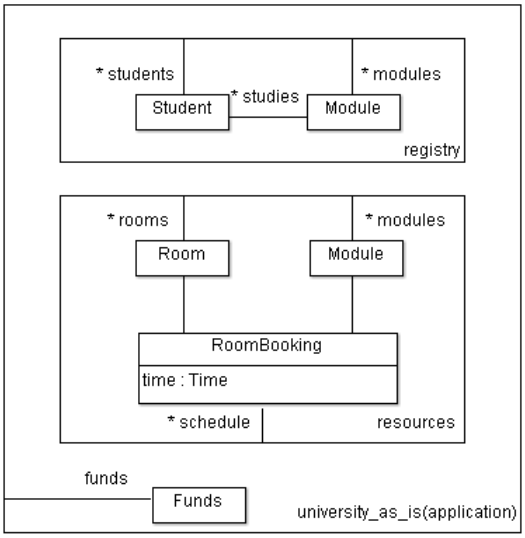

Figure 12: As Is (Application)

```
context university_as_is(business) inv:
  students.studies->subset(modules) and
  schedule->foraAll(s |
    rooms->includes(s.room) and
    modules->includes(s.room))
```

The operations are defined using OCL; each operation is given a context (the owning component), a name, some arguments, and a pre and post condition. The conditions specify the change in state that occurs as a result of performing the operation. The following example shows how student registration is defined (other operations are similar and are omitted.):

```
context university_as_is(business)::register(s:Student,m:Module)
post: students->includes(s) and
      modules->includes(m) and
      student.modules->includes(m)
```

4.2 Business to Application Refinement

A refinement to the diagram in figure **??** produces an *as-is* model of UME that describes the *application-layer*. The result of the refinement is shown in figure **??** where two components have been identified. The `registry` component is responsible for maintaining a database of students and their modules. The `resources` component is responsible for

maintaining a database of room bookings. The refinement also re-allocates some of the operations. This can be for a variety of reasons, for example operations may be reusable if they are decomposed or may map onto the technology layer more readily. For example;

```
context university_as_is(application)
  ::registerStudent(s:Student)=registry.registerStudent(s)
  ::registerModule(m:Module)=registry.registerModule(m)
  ::allocateStudent(s:Student,m:Module)=registry.allocateStudent(s,m)
context university_as_is(application)::registry
  ::registerStudent(s:Student)) post: students->includes(s)
  ::registerModule(m:Module) post: modules->includes(m)
  ::allocateStudent(s:Student,m:Module)
    post: s.modules->includes(m)
```

The refinement is complete when there is a refinement mapping between the business layer and the application layer. The mapping is defined as a sub-class of EA_refinement and is expressed in OCL as follows:

```
refine university_as_is(business,application) components :
  from.students = to.registry.students and
  from.modules = to.registry.modules and
  from.rooms = to.resources.rooms and
  from.modules = to.resources.modules and
  from.schedule = to.resources.schedule and
  from.funds = to.funds
```

The component mapping requires all of the application classes to map onto their business counterparts. Note that although the class `Module` appears twice in the application layer, the refinement requires both occurrences to be the same through the mapping to the single occurrence in the business layer. Note also that a student's modules and room bookings must be consistent in the application layer.

The refinement must define how each of the operations in the business layer is implemented in the application layer. Often business layer operations will have simply been allocated to an application layer component. In that case the refinement mapping is simple. In other cases the business layer operation may be implemented by a number of coordinating application layer operations as shown below:

```
refine university_as_is(business,application) operations :
from.register(s,m) =
  to.registerStudent(s);
  to.registerModule(m);
  to.allocateStudent(s,m)
```

4.3 Verification of the Refinement

In order to show that the *as-is* refinement is correct we must satisfy the condition that all semantic models satisfying the application layer also satisfy the business layer under the specified refinement mapping. If the refinement mapping is applied to the business layer constraints they become additional constraints at the application layer:

```
context university_as_is(application) inv:
  registry.students.studies->
    subset(registry.modules->union(resources.modules)) and
  resources.schedule->forAll(s |
    resources.rooms->includes(s.room) and
    resources.modules->union(registry.modules)->includes(s.room)) and
  resources.modules = registry.modules
```

Clearly any application layer trace that satisfies this constraint will map back to a business layer trace that satisfies the original business layer invariant.

The operations must also be shown to be valid. Consider a business layer step that involves registering a student. We must show that, when the refinement mapping is applied,

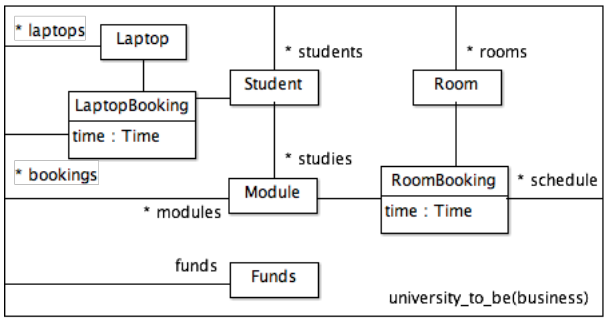

Figure 13: To Be (Business)

the resulting application trace maps back correctly to the business layer.

```
(Trace)[
  steps=[
    (Step)[pre=map-1(u);post=map-1(u'');
           message=(Message)[
             name='registerStudent';target=u;args=[s]]],
    (Step)[pre=map-1(u');post=map-1(u'');
           message=(Message)[
             name='registerModule';target=u';args=[m]]],
    (Step)[pre=map-1(u'');post=map-1(u''');
           message=(Message)[
             name='allocateModule';target=u'';args=[s],m]]]
     when u'.registry.students->includes(s) and
          u''.registry.modules->includes(m) and
          u'''.registry.students
             ->select(o | o.id = s.id).studies
             ->exists(o | o.id = m.id)
```

Applying the mapping `map` to this trace produces a business step as follows:

```
(Step)[
  pre=u
  post=u'''
  message=(Message)[name='register';target=u;args=[s,m]]]
    when u'''.students->includes(s) and
           u'''.modules->incudes(m) and
           u'''.students
             ->select(o | o.id = s.id).studies
             ->exists(o | o.id = m.id)
```

as required. Therefore the refinement is valid with respect to the *as-is* business to application refinement.

4.4 To Be Construction

Having performed an *as-is* analysis, UME constructs a *to-be* collection of models. As described by ArchiMate and other EA methods, the construction starts with a business layer that describes the essential features of the desirable state of the organization. For UME this is shown in figure **??**. In practice this probably starts with the model for the *as-is* business layer and modified it as required; however this is not necessary, the analysis could start from scratch. In the case of UME we add in two new concepts: `Laptop` and `LaptopBooking`. New operations are added:

```
context university_to_be(business)::registerLaptop(l)
  post: laptops->includes(l)
context university_to_be(business)::bookLaptop(s,l,t)
  pre: students->includes(s) and laptops->includes(l)
  post: bookings->exists(b |
    b.time = t and b.laptop = t and b.student = s)
```

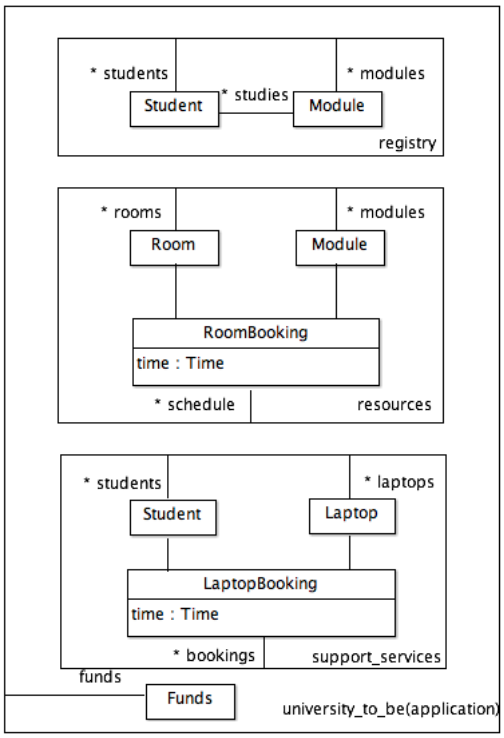

Figure 14: To Be (Application)

As before, the business layer is refined to produce the application layer as shown in figure **??**. Again, this is based on the *as-is* application layer, however a new component has been introduced to manage the lap-top bookings. The refinement is much as before:

```
refine university_to_be(business,application) components:
  // as before...
  from.laptops = support_services.laptops and
  from.students = support_services.students and
  from.bookings = student_services.bookings
```

We omit the operation refinement since no new features are introduced. The validation of the refinement is straightforward.

5. BUSINESS GOALS

Having constructed the *to-be* models based on modifications to the *as-is* models UME is now in a position to describe their business goals as described in the proposed extension to ArchiMate [**?**]. However, unlike the proposal, LEAP can be precise about the goals and can verify that they are satisfied by the architecture models. Furthermore, because is a semantics-based model driven approach, it is possible to formally check whether the goals are inconsistent. This section describes how this is achieved by specifying a simple business goal that is composed of sub-goals, showing how analysis leads to the detection of a problem with the business goals that can be fixed by a simple modification.

5.1 Precise Description

UME wishes to ensure that every student has access to a lap-top. Lap-tops are purchased at the start of the academic year out of UME funds. To simplify the example, we will assume that the only UME income is student tuition fees. Firstly, the university funds are expressed as a constraint (using tuition fees and laptop cost constants):

```
context university_to_be(business) inv:
  funds = students->size * tuition_fees -
          laptops->size * laptop_cost
```

The business goal is expressed in terms of the maximal set of students that can study at the same time. To calculate this set, we will assume the existence of an OCL function `sameTimes(T)` that returns the set of all sets of instances of `T` that have the same time (this assumes that type `T` has an attribute `time`). We also assume the existence of a function `pow(T)` that returns the set of all sets of instances of `T`. Clearly, `sameTimes` can be implemented in terms of `pow`.

The following invariant calculates `maxRooms` being the maximal set of rooms that can be booked at the same time, and calculates `maxStudents` being those students that should be in one of `maxRooms` because they study the module being taught there. The invariant constraint requires that UME does not go into deficit and that there are exactly sufficient lap-tops for the maximum number of students that must concurrently attend lectures:

```
context university_to_be(business)::maxRooms() =
  sameTimes(RoomBooking)->select(R |
     not sameTimes(RoomBooking)->exists(R' |
       (R->(union(R'))->subset(bookings) and
       R'->size > R->size)
context university_to_be(business)::maxStudents() =
  pow(Student)->select(S |
     S->subset(students) and
     S->forAll(s |
       s.studies->forAll(m |
         maxRooms()->exists(b |
           b.module = m))))
context university_to_be(business) inv:
  funds > 0 and laptops->size = maxStudents()->size
```

Using the extension to ArchiMate in [**?**], the business goal must be expressed using free text. This makes the goal open to interpretation and almost impossible to measure accurately. Using the LEAP approach to EA, the goals are expressed formally in OCL; together with the semantics for the LEAP language, this makes measurement and analysis very accurate. For example, the invariant above is expressed in terms of the business layer. The application and technology layers are defined as a result of applying precisely defined refinements to the business layer and therefore can be applied to the business goals. Having done this, the semantics can be used at lower levels of refinement to construct models of, say, the application or technology layers and to test whether the business goals have been achieved.

5.2 Inconsistency

This section provides a simple example that shows how the semantics can be used to check the consistency of business goals. We limit ourselves to the business layer, however the same process can be applied to refined models. UME has a business driver that is defined in the previous section: its funds must always be positive and its reputation must improve through making a lap-top available on-loan to all students. The relationship between the UME income and expenditure involves student and lap-top numbers and without a precise model it is not possible to check whether the UME goals are achievable.

Our business goal requires that no semantic trace leads to a violation of the invariant. The analysis is in two parts:

(1) Can we construct a component instance that violates the invariant? (2) Is the component instance part of a valid semantic trace? Suppose that we are given the following constants:

```
tuition_fees=1000
laptop_cost=2000
```

we can construct an instance of `university_to_be(business)` where

```
maxStudents=750
students=1000
```

The model describes how to calculate funds given the information above; the result is `-500000` which violates the invariant. The second part requires us to construct a chain of messages that leads to this state. This is easy since there is no limit on the number of students that can register at UME and there are no constraints on the number of concurrent room bookings; therefore there are many semantic traces that produce this state.

5.3 Strengthening the Conditions

The previous section has shown that a precise modelling approach to EA allows us to analyse the business goals for potential inconsistencies. We have seen that there are semantic traces that lead to inconsistent states. Since this is specified in the business layer and all other layers are linked via precisely modelled refinements, we know that *however* the business is architected, providing it satisfies the business layer, the architecture will not satisfy the goals.

The next step is to modify the business layer in order to ensure that the goals are satisfied by *every* semantic trace. This will guarantee that any refinement will satisfy the goals. Let's assume that the schedule is set before students register at UME. Since the number of students that will register is unkown at this point, some arbitrary scheduling is used to allocate modules to rooms (perhaps based on UME the previous year). Each time a student is registered (using the `register` operation, the funds available to UME can be calculated via the income from the new student and any new lap-top purchases.

As registration progresses (i.e. the semantic trace extends) one of two things will happen. Either the start of term will occur and the UME funds will have remained positive throughout the trace, or the funds will turn negative because room booking is such that too many lap-tops will be needed. In the former case no action need be taken. In the latter case UME needs to reschedule room bookings in order to try to reduce the number of required lap-tops. However, this will not always be possible because the number of available lecture rooms is fixed.

The specification of `register` is modified to accommodate these requirements. The pre-condition is modified to check that it is possible for the funds to remain positive after the student is registered for the module. The post-condition simply requires that the funds are still positive. Note that the post-condition does not mention the schedule and therefore allows the schedule to have changed; which may have been necessary to ensure that the university remains solvent:

```
context university_as_is(business)::can_reschedule(m)=
  let current_time = schedule.select(b | b.module = m).time
  in Time.allInstances->exists(t |
        rooms->exists(r |
          not schedule->exists(b | b.room = r)) and
        schedule->select(b | b.time = t)->size <
        schedule->select(b | b.time = current_time) - 1)

context univeristy_as_is(business)::register(s:Student,m:Module)
  pre: maxRooms()->exists(b | b.module = m) implies can_reschedule(m)
  post
    students->includes(s) and
    modules->includes(m) and
    s.students->includes(m)
```

The query operation `can_reschedule` returns true when there is a time where a room is free where the number of concurrent room bookings for the new time is less that that for the module `m`. By adding this as an extra pre-condition to the operation `register`, UME will stop recruiting students when it cannot reschedule modules in order to reduce the lap-top purchasing requirements. Given the business goal, room bookings will be modified as required.

Note that the strengthened business model implies that all layers that are refinements will also cause the rescheduling to occur. However, unlike the business layer, the application and technology layers will need to implement inter-component messages so that they co-ordinate in order that the pre-condition is met. Hence, a rather abstract business goal has potentially huge architectural implementations at lower levels. Since LEAP requires the goals to be formally specified, these can be checked throughout all the layers.

6. CONCLUSION

This paper has described Enterprise Architecture and provided an overview of a leading technology used to build EA models: ArchiMate. Archimate takes a layered approach to EA where an organization is modelled at business, application and technology layers. However, Archimate does not formally express the properties of the model elements and does not formally connect the elements between layers. It has been argued elasewhere that ArchiMate is weak in terms of Business Motivation Modelling, and a proposal has been made for adding a new motivation layer to ArchiMate models. The proposal uses the OMG BMM approach to business motivation where goals are expressed using free-format text. The proposal claims to be able to use goals expressed in this way to detect inconsistencies in an EA model and thereby show that an architecture will not satisfy an organization's business drivers.

This paper has described a model driven approach to AE that is based on language engineering. In such an approach, AE is considered as a domain and a language is described in terms of its syntax and semantics. This has been used to define a simple language called LEAP that captures the essential features of ArchiMate including the proposed business motivation extension. Since LEAP is based on formal modelling, all features have a semantics and ArchiMate layers are captured using formally modelled refinements. LEAP provides (1) orthogonal concepts; (2) precision through a semantic domain; (3) precisely modelled refinement relationships. Unlike ArchiMate, there is no limit to the number of layers in LEAP and each layer is expressed in the same language (albeit at a different level of detail). Since goals are captured as OCL constraints over LEAP models they can be formally checked for consistency. A simple example involving a student lap-top loan scheme has been used to explain the key features of LEAP.

LEAP is intended to show how a language driven approach to modelling can be used to add precision to EA technologies such as ArchiMate. As discussed earlier, there

is a straightforward mapping between LEAP and ArchiMate based on a UML profile. In terms of EA methods, LEAP could be used to add precision to existing ArchiMate models or ArchiMate could be used to document LEAP-based models. There are many tools that support UML-style modelling including tools that execute OCL constraints. LEAP allows these tools to be used for EA.

Other model driven approaches to AE include SEAM [**?**] and the Model Driven Architecture (MDA) initiative from the OMG. However, none of these approaches are aligned with the leading technologies such as ArchiMate and none are precise.

LEAP has been used in this paper to show that the proposed business motivation extension to ArchiMate can be supported in a precise way that allows formal analysis of the business goals. LEAP has also been used to analyse business change by precisely modelling a university *as-is* and *to-be*. We claim that LEAP can form the basis of other EA analyses such as quality; compliance and mergers. LEAP can form the basis of a family of EA *domain specific languages* that provide specific support for the precise expression of models required to undertake the analysis. This is an area for future work.