

Relating AO Requirements to AO Architecture

R. Chitchyan¹, M. Pinto², L. Fuentes², A. Rashid¹

Abstract - We have identified a gap in relating AO requirements engineering and AO architecture design: a large amount of important information generated during the mapping of requirements to architecture is lost in the final representation of the architecture. In this work our primary aim is to outline a traceability schema that will provide support for recording such information generated during the mapping process. Here we focus on the mapping from requirements to architecture though the schema could be used for relating other development stages.

Index Terms—aspect-oriented requirements engineering and architecture design, traceability, mapping requirements to architecture.

I. INTRODUCTION

By now a significant body of work has been carried out for Aspect-Oriented Requirements Engineering (AORE) [1-4] and Architecture Design (AOAD) [5-7]. However, up till now these two areas have been investigated largely independently. Although some research [2, 8] has pointed out that the linking of the AORE and AOAD is desirable, there has been no extensive work carried out in to link AORE and architectural analysis.

In the present paper we provide a first attempt to fill in this gap in relating AORE and AOAD. Since there has been no consensus on what AORE and AOAD are, we first clarify the use of these terms in our work.

We define Aspect-Oriented Requirements Engineering as the requirements engineering approaches which address identification, representation, and treatment of all types of requirements (functional, non-functional, localised, and broadly scoped) and their mutual influences. The novel perspective contributed by of AORE is the systematic addressing of crosscutting concerns (aspects) and their influences. In [8] (section 3.4) we have discussed that the emergence of AORE approaches has been motivated by lack of composability and traceability in the traditional RE work, as well as by the need to support the early stages of software development for the newly emerging Aspect-Oriented Programming techniques.

Correspondingly, Aspect-Oriented Architecture Design is defined as the architectural stage of software development techniques where all the requirements delivered by the AORE,

as well as the newly identified architectural issues, are explicitly represented and treated in respective architectural artefacts. In difference with traditional architectural approaches, AOAD explicitly treats aspects. Extending the discussion presented in [1], we expect that an aspect can map from requirements to architecture as follows:

- Architectural Component: a localised architectural module or an element of a module (e.g., method for composition correctness checking);
- Architectural Aspect: an architectural module that has a broad influence on a number of other modules (e.g., security module, providing authorisation, authentication and encryption/decryption functionality);
- Architectural Decision: a localised decision for a particular architecture;
- Aspectual Architectural Decision: decision taken under the influence (considerations) of the crosscutting concerns (e.g., use of semi-centralised peer-to-peer topology to provide availability).
- Non-Architectural Decision: decisions related to business process review, staffing, etc (e.g., decision to hire a security guard).

In the rest of this paper we present why and how we provide support for relating AORE and AOAD through use of a traceability schema. We believe in the future this traceability approach can be extended to fully support traceability throughout the development.

Currently we present an initial outline of part of the schema. The following section presents a motivating example, explaining why a traceability schema for requirements to architecture mapping is required. Section 3 then describes the schema itself. Section 4 outlines the applicability of this approach to the Pet Shop example required by the workshop. Finally, section 5 presents conclusions.

II. MOTIVATING EXAMPLE

Our initial objective was to establish an example of continuity between AORE and AOAD by closely linking the artefacts produced in the Requirements Engineering and Architecture lifecycle phases. For this we selected a particular AORE approach: that discussed in [1] which has a Requirements Description Language (RDL), and a particular AOAD approach: DAOP-ADL [5], which is an Architecture Description Language (ADL). To link the respective requirements and architecture artefacts we used the RDL artefacts for the Portuguese Toll Gate System as described in [1] (see case study briefly outlined below) and mapped these

R. Chitchyan and A. Rashid are with the Computing Department Lancaster University, Lancaster, LA2 4WA (e-mail: {rouza,marash}@comp.lancs.ac.uk).

M. Pinto and L. Fuentes are with Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, Málaga, Spain (e-mail: {pinto,lcc}@lcc.uma.es).

to the ADL descriptions of the DAOP-ADL.

Our selected RDL provides representations for concerns, viewpoints and composition specifications. The concerns represent broadly scoped properties (e.g., security, compatibility in Fig. 1). Viewpoints present a certain perspective on the system (e.g., vehicle's perspective). Composition specification defines how concerns and viewpoints combine together into amalgamated set of requirements.

```
<?xml version="1.0" ?>
<Concern name="Compatibility">
  <Requirement id="1">
    The system must be compatible with systems used to:
    <Requirement id="1.1">activate and reactivate gizmos;</Requirement>
    <Requirement id="1.2">deal with infraction incidents;</Requirement>
    <Requirement id="1.3">charge for usage;</Requirement>
  </Requirement>
</Concern>
```

Fig.1. Requirements of the Compatibility concern

The DAOP-ADL provides constructs for aspectual (e.g., security component) and non-aspectual (e.g., police system) components and their interfaces, and specifications for composition of aspectual and non-aspectual components.

The briefly summarized description of the Portuguese Toll Gate System [1] follows. In this system drivers at certain roads are charged a fee at toll gates automatically. Authorised vehicles can pay electronically whereby the Toll Gate System directly interacts with the Debiting System (DS). Non-authorized vehicle owners receive a fine to their address. In order to be authorised, a driver has to install a device (a gizmo) in his/her vehicle. Gizmo is activated via Auto-Teller Machines (ATM). The gizmo is read by the toll gate sensors that use the information to debit the respective amount. Also Police System (PS) monitors the legal use of vehicles passing through the toll gate.

Through this mapping exercise we discovered that many of the artefacts described in our RDL could simply correspond to the DAOP-ADL artefacts, as summarised in Table 1.

TABLE I.
SUMMARY OF RDL AND ADL CORRESPONDING ELEMENTS

RDL	DAOP-ADL	Comments
Concerns, viewpoints, requirements	Components, Decisions	Components encompass concerns and viewpoints
Aspectual concerns	Aspects, Architectural Decisions	Aspects are components with special interfaces
Composition Rules	Component and Aspect Composition Rules	DAOP-ADL compositions define correspondence between required/expected interfaces of components and aspects
Composition Operators	Types of Join points, Decisions	The RDL operators that imply a temporal sequence map onto before/after joinpoints, the concurrency semantics (e.g. with operator) maps to a concurrency tag, etc.

However, it also came to our attention that a large amount of important information is generated during the actual mapping process, but lost in the final graphical (boxes-and-lines) or ADL representation of architecture.

For instance, in many cases the mapping is not singular: many alternatives are available for grouping concerns into components, or aspects; some requirements level artefacts (e.g., availability) do not result in modules, but in architectural decisions (e.g., selected topology); others do not receive any physical representation in architecture but may prompt the architect to suggest some implementation-related information etc.

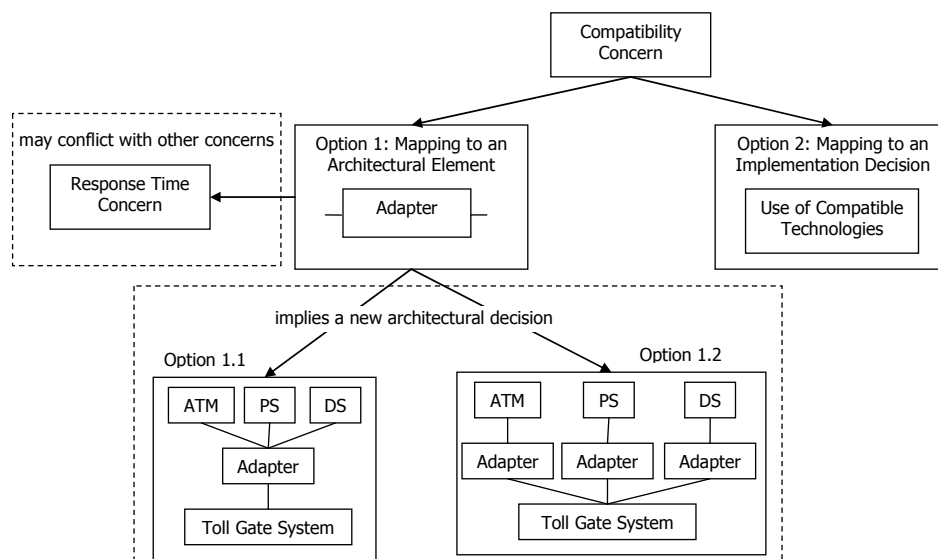


Fig. 2. Mapping of the Compatibility Concern From Requirements to Architecture

We illustrate these issues through the mapping from requirement to architecture of the compatibility aspect in the toll gate example the RDL description of which is illustrated in Figure 1 (reused from [1]).

Compatibility in the toll gate example may be mapped from an RDL description of concern to an architectural element in ADL, as shown by Option 1 of Figure 2. But it could also be mapped to an implementation decision (Option 2 in Figure 2). These options depend on the decisions taken by software architects.

On the one hand, in option 1 the software architect takes the decision of mapping the compatibility concern to an adapter on basis that ATMs, the Police System and the Debiting System are pre-existing subsystems, with their own hardware, software and interaction policies (i.e. they are considered COTS components). Therefore, the toll gate system must work with them without changing them. This decision implies better flexibility and adaptability; though it can penalise the response time of the toll gate system.

On the other hand, in option 2 the software architect decides to postpone the achievement of the compatibility concern to later phases of the development. Concretely, he/she decides that compatibility would be achieved by integrating the implementation of the toll gate system with the current software of existing subsystems. This decision will result in less adaptable systems, though with probably better performance than previous option.

Thus, requirements can be mapped to alternative choices relating to architecture or implementation, but only architectural choice could be recorded by the ADL. The implementation-based choice (if selected) will remain implicit to the ADL.

Not only requirements influence different levels of the software lifecycle (e.g., architecture or implementation), but they can be mapped to different kinds of software artefacts at a given level at one time. As demonstrated in Figure 2, once an architect decides to map compatibility to an adapter element, he/she must also decide on the structure of the architecture (Options 1.1 or 1.2 in Figure 2).

We found that it was not possible to modularly represent the decisions on architectural element (e.g., adapter) and topology (Options 1.1 or 1.2) as architectural representations of the compatibility aspect. Moreover, the justifications for opting either for a single- or multiple-adapter topology would not be recorded in the ADL, and would be a lost resource for evolution¹.

Also, there was no mechanism to pass other information that did not become part of the ADL model (e.g., desirable implementation techniques, notes of possible complications, reasoning for architectural decisions, etc.) from requirements to architecture and further on to implementation etc. stages¹.

For instance, as shown in Figure 2, having decided on use of adapters, an experienced architect will be aware that such architectural choice will conflict with the response time requirement. In this case, response time is an example of concern which will not be realised at architecture or design stage, but needs to be flagged as an issue to be addressed at implementation time. However, the architect has no easy way of instructing the designers and developers that adapter implementation must be very efficient and is related to response time issues.

Besides, we realised that in the mapping process there was no forward and backward traceability support. For instance, the inclusion of the adapter at the architectural level may force to review the requirements of the “Response Time” concern. However neither the reasons of revision nor the previous version of requirements will be retained in RDL or ADL, denying the option of roll-back or review of the change.

The importance of the above described information for the later phases of the development is obvious: it will help to justify the selected architecture, shape design, help to pass relevant knowledge down to the implementation, and explain decisions and available options for future evolution and maintenance. However, we found that the existing approaches, including the RDL and the DAOP-ADL languages used in our study, do not include support for recording this information, which is, consequently, lost for the later stages. Thus, due to the importance of such information we felt the need to provide such a support.

III. INITIAL OUTLINE OF THE TRACEABILITY SCHEMA

We propose to provide such a support through a traceability mechanism realised via traceability framework. This framework includes the schemas of all lifecycle artefacts and allows to record mappings to different elements (e.g., architectural components, aspects, decisions, etc.) as well as the supporting information related to both the artefacts themselves and their mapping. Such traceability support could be provided for all artefacts and transitions between all distinct stages of lifecycle, as well as for the evolutionary change. However, in the present paper we demonstrate only a part for Requirements to Architecture mapping.

An outline for such a mapping schema is demonstrated in Figure 3. Here the combination of RDL, DAOP-ADL, and mapping and description for the compatibility concern are developed as separate schemas, but they all are contributing elements to the Traceability framework. While the RDL and DAOP-ADL describe the requirements and architectural elements (e.g., concerns, requirements and components, etc.) correspondingly, the mapping and decisions schema provides details of how each requirements element is mapped to (<mappedTo> tag) the architectural elements.

¹ While the lack of support for such information representation is an obvious flaw of our chosen RDL and ADL, this is a common characteristic of present day description languages which focus on describing only the artefacts, but not the detail of obtaining them, or modularly representing the crosscutting concerns.

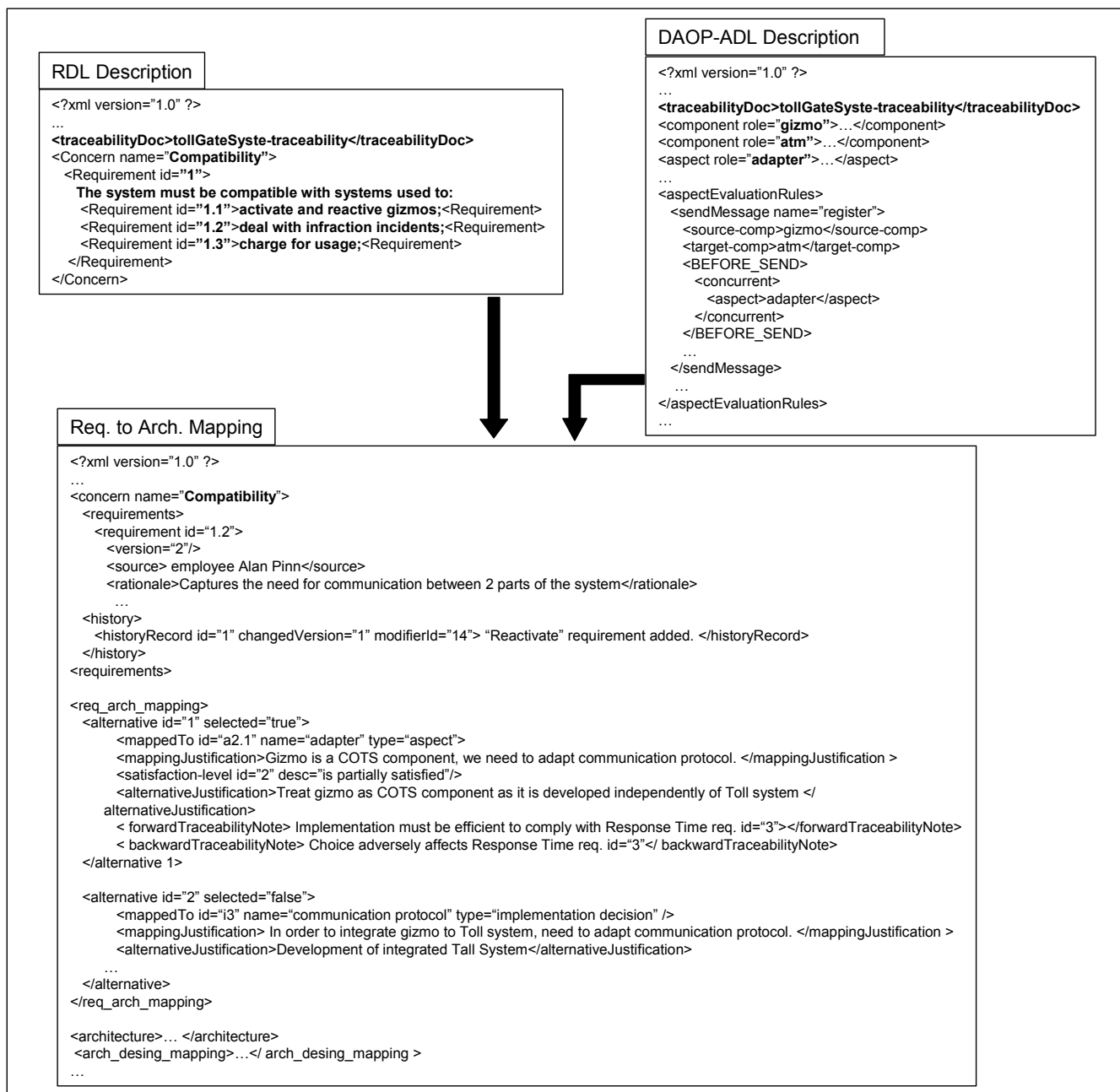


Fig. 3. Traceability framework: part for requirements to architecture design mapping

(Note: definitions of adapter (referred to via id a2.1) and communication protocol (referred to via id i3) are not included).

Moreover, the possible alternative (<alternative> tag) mappings and decisions are also recorded and change history (<history> tag) for each artefact is maintained.

From the realisation perspective, the framework can be composed of separate schemas for:

- Descriptions of requirements, architecture, design, implementation and maintenance artefacts (e.g., RDL, DAOP-ADL, etc.);
- Mapping and decisions (e.g., architectural, and non-architectural);
- Evaluation models;
- Accompanying notes, etc.

The initial outline of part of the traceability schema for

Requirements to Architecture mapping elements is presented in Table 2.

It could be important to point out that using such a traceability schema does not enforce any particular development process or mapping sequence. However we are of the view that it is not feasible to consider the architecture for one-concern-at-a-time since each concern in isolation will be suggesting a different architecture [2]. Thus, it could be appropriate to use a "Twin Picks"-like approach [9] simultaneously deriving architecture and requirements as well as populating the traceability schema.

Having the framework as a reference point, the developers at any stage of the software lifecycle will be

TABLE II
INITIAL OUTLINE OF INFORMATION IN THE TRACEABILITY SCHEMA FOR
REQUIREMENTS TO ARCHITECTURE MAPPING

Requirement Level	Mapping to Architecture Level
RE Element 1. Concern 1.1. Aspectual Concern 1.2. Non-Aspectual Concern 2. Viewpoint 3. Requirement	Architectural Elements 1. Architectural Element (AE) 1.1. Aspectual AE (aspect) 1.2. Non-Aspectual AE 2. Architectural Decision (AD) 2.1. Aspectual AD 2.2. Non-Aspectual AD 3. Non Architectural Decision (e.g. monitoring for requirement compliance)
Weight: the relative weight allocated to a requirement by a stakeholder	Level of Satisfaction 1. is satisfied (at architectural level) 2. is partially satisfied 3. is not satisfied
Influence: the set of other elements influenced by (positively or negatively contributed towards) this element	Consequences of the Influence 1. is propagated to 2. add new requirement 3. add new decision 4. depends on (other decisions, requirements, etc.) 5. influences (other decisions, requirements, etc.) 6. implies
Emergent Requirements Requirements identified at architecture design (or other) stage, including links to the decisions that motivated their inclusion/ consideration (if appropriate).	Links to Emergent Elements Links to the decisions in other stages of the development that were propagated from architecture, and to the modifications in the requirements that were motivated by a decision of the software architect.
Composition Elements Operators Actions ...	Composition Elements Joinpoints or decisions (e.g. concurrency decisions) AE interfaces or decisions ...

able to obtain and pass to each other the relevant information, as well as be informed about the decisions and choices taken by others.

IV. DISCUSSION OF SCHEMA APPLICABILITY FOR THE PET SHOP EXAMPLE

For the benefit of the Early Aspects 05 (OOPSLA) workshop, we very briefly outline the applicability of the above discussed traceability schema to the Pet Shop example [10]. If it is intended to apply aspects for this example development, the above presented compatibility concern example could arise for the Shop's integration with the Credit Card Payment System. In this case similar requirements to architecture mapping options will require consideration and some reasoning similar to that discussed earlier will be applied.

However, even when considering the provided pure OO implementation, we immediately identify the whole wealth of "assumed" knowledge for requirements to architecture mapping which will be lost if not recorded in a traceability

schema. Such "assumed knowledge" examples, vital for evolution, are, for instance, decisions on model-view-controller vs. workflow process architecture selection; web-centric vs. EJB design, or justification of the level of distribution for a given component and the effect of the selected options on such requirements as availability and response time, etc.

V. CONCLUSIONS

In this paper we have demonstrated the need of and first outline of a comprehensive traceability framework both for recording of information generated in the process of artefact mapping between development stages (using Requirements to Architecture mapping example), and for passing information between development stages, as well as modular reasoning preservation.

Our current work is focused on elaborating and improving this framework (e.g., introducing constructs for pointcuts into schemas, making schema structure more suitable for automated processing, etc.). Once completed, the framework will serve as a back-end for an AOSD process tool (with a graphical user interface to hide the XML tags and assist usability) for supporting seamless AO development (initially from Requirements to Architecture and Design).

Another area of our future work lies in assessing the expressiveness of our traceability language and improving it in order to ensure that the language will support representation of all our desired traceability characteristics.

We also realise that populating mapping schemas will be a time-consuming activity, and we intend to automate population of directly linked fields (e.g., allocation of requirement elements to components, etc.). However, in cases where mapping is not obvious and additional information passing and recording is required automation will not be possible. Nevertheless, as discussed in the motivating example above, such information is essential for successful development and we believe its benefits (for development and evolution) will outweigh the extra time costs.

ACKNOWLEDGMENT

This work is supported by European Commission Grant IST-2-004349: European Network of Excellence on AOSD (AOSD-Europe).

REFERENCES

- [1] A. Rashid, A. Moreira, and J. Araujo, "Modularisation and Composition of Aspectual Requirements," presented at 2nd International Conference on Aspect Oriented Software Development (AOSD), Boston, USA, 2003.
- [2] A. Moreira, J. Araujo, and A. Rashid, "Multi-Dimensional Separation of Concerns in Requirements Engineering," presented at Requirements Engineering Conference (RE 05), Paris, France, 2005.
- [3] J. Whittle and J. Araujo, "Scenario Modeling with Aspects," *IEEE Proceedings - Software*, vol. 151, pp. 157-172, 2004.
- [4] E. Baniassad and S. Clarke, "Theme: An Approach for Aspect-Oriented Analysis and Design," presented at International Conference on Software Engineering, Edinburgh, Scotland, UK, 2004.

- [5] M. Pinto, L. Fuentes, and J. M. Troya, "DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development," International Conference on GPCE, Erfurt, Germany, 2003.
- [6] U. Kulesza, A. Garcia, and C. Lucena, "Towards a method for the development of aspect-oriented generative approaches," presented at Workshop on Early Aspects (held with OOPSLA 2004), 2004.
- [7] U. Kulesza, A. Garcia, and C. Lucena, "Generating aspect-oriented agent architectures," Workshop on Early Aspects (held with AOSD 2004), 2004.
- [8] R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, M. Pinto, J. Bakker, B. Tekinerdogan, S. Clarke, and A. Jackson, "Survey of Analysis and Design Approaches," Lancaster University, Lancaster, Survey Report AOSD-Europe-ULANC-9, 2005.
- [9] B. Nuseibeh, "Weaving Together Requirements and Architectures," *EEE Computer*, vol. 34, pp. 115-117, 2001.
- [10] Web Site: *Pet Shop Example in "Designing Enterprise Applications with the J2EE Platform", Second Edition*, http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/index.html, Sun Developer Network, 2005.