Comparing architectural styles for distributed expert knowledge modules in intelligent tutoring systems

Master thesis

Author: Gideon Teeuwen
Student number: 851132030
Date of defense: 25 August 2016
Comparing architectural styles for distributed expert knowledge modules in intelligent tutoring systems

Master thesis

Graduation committee: prof. dr. Johan Jeuring (chairman)
                        dr. Bastiaan Heeren (supervisor)

Institute and faculty: Open University of the Netherlands
                        Faculty of Management, Science & Technology

Program and course code: Master Software Engineering
                         IM9906
Acknowledgements

I would like to thank dr. Bastiaan Heeren and prof. dr. Johan Jeuring for reviewing this thesis and for giving their useful feedback on my research to architectural styles for distributed intelligent tutoring systems.

The image used for the cover illustration comes from *Furniture Design Ideas*, http://furn.wonderenergybar.com/.
Contents

Abstract .......................................................................................................................................................................................................................... 3
1. Introduction ................................................................................................................................................................................................................. 4
2. Related work .................................................................................................................................................................................................................. 5
  2.1 Intelligent tutoring systems (ITSs) .......................................................................................................................................................... 5
  2.2 IDEAS framework ....................................................................................................................................................................................... 7
  2.3 Architecture ............................................................................................................................................................................................. 7
3. Research incentive, scope and contribution .................................................................................................................................................. 8
  3.1 Research incentive .................................................................................................................................................................................... 8
  3.2 Scope .............................................................................................................................................................................................................. 9
  3.3 Research goals and research questions ............................................................................................................................................... 9
  3.4 Research contribution ........................................................................................................................................................................... 10
4. Architectural quality attributes for ITSs ....................................................................................................................................................... 11
  4.1 ISO 25010 ................................................................................................................................................................................................ 11
  4.2 Quality for ITSs ...................................................................................................................................................................................... 13
  4.3 Distributed expert knowledge modules ............................................................................................................................................. 13
  4.4 Stakeholders and concerns ............................................................................................................................................................... 13
  4.5 Quality attributes for distributed ITS environments ...................................................................................................................... 15
  4.6 Evolution of ITSs .................................................................................................................................................................................. 18
5. Scoring architectural styles on the quality attributes important for distributed ITSs .................................................................................. 19
  5.1 REST architectural style ..................................................................................................................................................................... 19
  5.2 REST quality aspects ........................................................................................................................................................................... 21
  5.3 Service oriented architecture with web services ............................................................................................................................ 23
  5.4 Quality aspects of the Service Oriented Architectural style ......................................................................................................... 25
  5.5 Plug-in architecture ........................................................................................................................................................................... 27
  5.6 Quality aspects of the plug-in architectural style ............................................................................................................................ 28
6. Comparison of architectural styles and distributed ITSs........................................................................................................................... 30
  6.1 Comparison of architectural styles ...................................................................................................................................................... 32
  6.2 The expert knowledge module as a distributed component ..................................................................................................... 33
7. Architectural views for distributed ITSs .................................................................................................................................................... 33
  7.1 Logical view ............................................................................................................................................................................................. 35
7.2 Resource view

8. Prototype

  8.1 Frameworks for building a prototype

  8.2 Expert knowledge module for simple calculations

  8.3 REST layer

9. Conclusion, discussion and future work

  9.1 Conclusion

  9.2 Discussion

  9.3 Future work

Bibliography
Abstract
Intelligent Tutoring Systems (ITSs) are computerized systems for learning-by-doing. These systems provide students with immediate and customized feedback on learning tasks. ITSs consist of modules that are connected to each other. The next step in the field of ITS development is the distribution of ITSs using the internet as a distribution medium. The distribution of ITS modules over multiple locations improves access to ITSs. This increase of ITS access gives the advantage that users from all over the world are able to benefit from education provided by those ITSs. This research focuses on the distribution of the ITS module that provides expert knowledge services. For the distribution of such an expert knowledge module we need to use an architectural style because this gives a standard interface, which increases the reusability and operability of the expert knowledge module. To provide expert knowledge modules in a distributed way we need to answer the research question: ‘How can we compare and evaluate REST, Web services and Plug-in architectural styles for the distribution of the expert knowledge module in an intelligent tutoring system?’.

We present an assessment method for selecting an architectural style. Using the assessment method on three architectural styles, we selected the REST architectural style as the style that best supports the distribution of expert knowledge modules. With this assessment method we also analyzed the trade-offs that come with selecting REST. This assessment method enables us to answer the research question.

We present a prototype and architectural views based on REST to demonstrate that the assessment method correctly scores REST as an appropriate architectural style for the distribution of expert knowledge modules.
1. Introduction

An intelligent tutoring system (ITS) [3, 13] is a technique for learning-by-doing. These computerized systems give immediate and customized feedback on student actions on learning tasks in, for example, the field of math or computer programming. ITSs can also steer the learning of students while taking the student’s knowledge level into account.

The architecture of ITSs has evolved from a linear architecture in the 50s to a general form of an ITS architecture consisting of a division into conceptual modules in the late 80s [3]. This general form is still used today for building ITSs [3] (see Figure 1). A next step in the evolution of ITSs is the distributed architecture where these conceptual modules are separated from each other and distributed over multiple locations (via the internet for example). The distribution of ITS modules over multiple locations improves access to ITSs. This increase of ITS access gives the advantage that users from all over the world are able to benefit from education provided by those ITSs.

There has already been done research on the deployment of distributed ITSs. A start is made by Alpert, Singley and Fairweather by deploying an ITS on the web [14]. This is done by separating the user interface module from the ITS environment.

To further increase the deployment over internet, it would be useful to distribute the expert knowledge services as well. In that case, ITS developers from all over the world can have access to several expert knowledge modules for building their own ITS.

The main contribution of this research is to find an architectural style for the distribution of the expert knowledge modules. Heeren and Jeuring have made a start with the REST architectural style [7] and consider addressing the REST principles in their domain reasoners as future work. To the best of our knowledge, there has not been done research on architectural styles for distributed expert knowledge modules.

The focus of this research will be on finding the quality attributes needed for comparing architectural styles, which existing architectural style to use, and which part of the expert knowledge module should be distributed. Another important point is which programming framework to use for building ITSs with a distributed expert knowledge module using the chosen architectural style.

The next chapter gives an overview about existing ITS research. It consists of a description of ITSs, the IDEAS framework for building domain reasoners and the architectural styles we will use in this research. Chapter 3 describes the motivation, scope and threats to validity for this research on architectural styles for distributed ITSs. Chapter 3 also presents the research question and the contribution of this research to existing work. Chapter 4 contains the analysis of ITSs with regard to software quality attributes. Chapter 5 gives a description and analysis of the set of architectural styles used in this research. In chapter 6 we describe how the architectural styles match the quality attributes of distributed ITSs. The REST architectural style turns out to match the quality attributes needed by distributed ITSs. Chapter 7 presents the architectural view of a distributed ITS using the REST style. In chapter 8 we explain how we build the prototype. In chapter 9 we give the results of our research and we highlight the points of discussion that can be used as input for future work.
2. Related work

2.1 Intelligent tutoring systems (ITSs)
One of the most effective ways of teaching is one-on-one tutoring [3, 14]. Unfortunately, the ideal situation that every student has his own private tutor is not feasible financially and logistically.

A technology that simulates this one-on-one tutoring is intelligent tutoring systems (ITSs) [3, 13]. ITSs consist of a so-called inner loop and outer loop. The difference between the inner loop and outer loop is that the first focuses on analyzing the solution of a task in a step-by-step way, whereas the latter focuses on selecting the appropriate task [13].

The main advantages of ITSs are:
- Aiding students with solving exercises by giving stepwise hints for solving those exercises.
- Providing feedback to the student about his solution to a task. Thus not only checking if the given solution is correct, but also if the correct steps have been taken for solving a task (inner loop). This feedback service tells the student how well he solved the task. For instance, the solution may be correct, but how efficient was the proposed solution? Was another solution more efficient?
- Selecting tasks according to the knowledge level of the student (outer loop) using a student model and a pool of exercises.

In the literature about computerized tutoring systems, multiple names are used for those systems. Computer aided instruction (CAI) [3], computer based training (CBT) and web-based homework (WBH) [13] are a few names used in the literature. The difference between an ITS on the one hand and CAI, CBT and WBH on the other is that the latter generally lack an inner loop. Systems that do have an inner loop are called ITSs [13]. Therefore, the definition of an ITS used in this thesis is: a tutoring system that consists of two loops: an inner loop and an outer loop.

The inner loop is designed to solve exercises in a stepwise manner as a human would solve exercises. Since there can be more than one way to correctly solve an exercise (a mathematical equation for example), the inner loop needs to have methods to analyze a student step in an exercise or to analyze all possible steps to give hints. In that way it is also possible for an ITS to recognize common mistakes and give hints about them: the so called buggy rules [7, 13].

For selecting an appropriate task for a student, an ITS needs to know what the knowledge level of the student is. For example: there is a difference in types and difficulty level of exercises for novice programmers who want to learn a programming language and experienced programmers who want to refresh their knowledge. This selection of tasks is the responsibility of the outer loop.

We can identify several responsibilities in an ITS: maintaining a pool of tasks, keeping track of student information, presenting tasks to students, etcetera. An important idea from an architectural point of view is the division of responsibilities into different modules or components [15]. There is a consensus about how an ITS should be divided into components with their own responsibility [3]. We can identify four components (see Figure 1):
- The student model module
  o Responsible for keeping track of student information such as their learning progress and level of expertise. In other words: the responsibility is to construct a profile of the student [3].
- The tutoring module
  o Contains the teaching strategy. It is closely linked to the student model and it orchestrates all pedagogic interventions [3].
- The user interface module
  o Responsible for the communication between the user and the intelligent tutoring system.
- The expert knowledge module
  o This module contains domain specific knowledge. This module can further be divided into two separate parts: the collection of exercises and a module that can reason about the problem domain. The latter is also called a domain reasoner [7]. The domain reasoner is responsible for the stepwise exercises and for providing the feedback to a student.

The arrows between the components depict the information that flows between the components. Note that Figure 1 is basic consensus about how an ITS should look in an architectural way. It is also possible to let an ITS use an external expert knowledge module that is not a part of the ITS itself [7]. Figure 2 shows the separation between an intelligent tutoring system and the expert knowledge module. The advantage is that multiple ITSs can share a common expert knowledge module. This also works the other way around: an ITS can use multiple expert knowledge modules. In Heeren and Jeuring [7], an ITS that uses an external expert knowledge module is considered to be a single entity called a learning environment. In this research we refer to a distributed ITS as an ITS that consists of a learning environment that uses an external expert knowledge module.
IDEAS framework

IDEAS is an abbreviation for ‘interactive Domain-specific Exercise Assistants’ and is developed by the Open University of the Netherlands and Utrecht University. IDEAS is a framework for developing domain reasoners that give intelligent feedback. Using IDEAS, domain reasoners can be developed for solving linear, quadratic and higher-degree equations, gaussian elimination, and many other domains [32]. Math-Bridge (a teaching environment for learning math) and Ask-Elle (a Haskell tutor) are examples of ITSs that use the IDEAS framework for their domain reasoning.

2.3 Architecture

The main goal of the master thesis is to find an architectural style for ITSs where the architecture consists of a learning environment and an external expert knowledge module. The learning environment consists of modules such as a student module, tutoring module and user interface module. The external expert knowledge module can be used by different learning environments. This external expert knowledge module provides feedback services for the stepwise exercises.

A software architecture is defined by the IEEE standard 1471: “The fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution”. In the situation of a distributed ITS: an architecture is a design of the distributed ITS as one system built with different components that are somehow connected to each other and the environment. The software architecture of distributed ITSs also states clear principles, for instance, “ITS users have access to the ITS via the user interface module”.

Figure 2. Learning environments using feedback services provided by domain reasoners (from Heeren and Jeuring [7]).
In this research an architectural style is considered a reusable architectural blueprint that has well-known characteristics that make it appropriate to use to satisfy particular types of requirements\(^1\) [15]. Gorton uses the word ‘architectural blueprint’ as a description of the structure and interaction between collections of participating components. An architectural style can be used to design the architecture of an ITS with a distributed expert knowledge module.

In this master thesis, the scope will be on the connection between learning environments and expert knowledge modules. Because the expert knowledge module is external to the learning environment, the architecture of an ITS must be a distributed architecture. The learning environment needs the feedback services from the external expert knowledge module. Therefore, there must be a way of communication between the learning environment and the expert knowledge module.

There are several architectural styles that can be used for connecting learning environments and the expert knowledge module in a distributed environment. In this research we will consider three architectural styles for distributed ITSs:

- REST architecture [7]
- Service oriented architecture with web services [16, 17]
- Plug-in architecture [14, 18]

The reason for considering these architectural styles is that these architectural styles already exist within e-learning environments or ITSs. However, as mentioned in the introduction, there has not been done research on architectures for distributed expert knowledge modules.

In chapter 5 a brief description of each of those architectural styles will be given and these styles will be weighed against software quality attributes. In other words: how do the architectural styles scores on these quality attributes. The scoring of the architectural styles will then be compared to the scoring of ITSs and the architectural style that fits best will be selected for constructing a proof-of-concept implementation.

3. Research incentive, scope and contribution

3.1 Research incentive
The rationale behind this research is the increase of reuse and interoperability of expert knowledge modules. Reuse and interoperability avoid duplication of effort since an ITS developer does not need to develop his own knowledge module if there is a suitable module that is already available. The availability of knowledge modules over the internet also increases the possibility for the internet community to improve and scrutinize the existing knowledge modules.

We have not found research on how the distribution of learning environments and expert knowledge modules can be done. In Heeren and Jeuring [7] there is made a start with separating learning environments from their expert knowledge modules in IDEAS (see Figure 2). However, Heeren and Jeuring did not develop an architectural style for this separation. The disadvantage of separating learning environments from expert knowledge modules without considering an architectural style is that there is no standardization in the communication between the learning environment and the expert knowledge modules.

\(^1\) Architectural styles are called ‘patterns’ in Gorton
module. In Figure 2 we see that each learning environment that wants to use a certain expert knowledge module needs to know which methods are defined in that expert knowledge module to get the feedback service. We need to develop a standard communication interface between those two components to get the reuse and interoperability advantage. Therefore we need this research on architectural styles to obtain this standardization.

3.2 Scope
Considering the limited time for this research, the scope will only be on the three architectural styles mentioned in Section 2.3 (REST, web services and plug-in architectural styles). The reason for this selection is that the plug-in style and web service styles are already used in existing ITSs for decoupling the user interface from the ITS [16, 17] and in animated simulations [18]. In the IDEAS framework there has been made a start with a REST architecture, but a complete architectural style for expert knowledge modules is considered as future work [7]. This limited set of architectural styles forms a threat to validity, because we might overlook an architectural style not mentioned in this research that scores better than the chosen architectural style. The scope of this research is therefore that it develops an assessment method for selecting an architectural style for the distribution of expert knowledge modules. The incentive of this research is to obtain the benefits mentioned in the first section of this chapter that come with a distributed ITS. This will be done by finding an architectural style that covers the quality attributes important for distributed ITSs. Finding the best suitable architectural style using techniques such as systematic literature review is considered future work. However, the assessment method used in this research can be of assistance for assessing architectural styles found in future work.

The focus of this research on ITSs will be on the decoupling of the expert knowledge model from the learning environment only. The decoupling of the user interface and the learning environment will not be investigated. There has already been done research in this field by, for example, Serman et al. [14], Westerkamp [16] and Pankratius et al. [17].

A prototype will be made as a proof of concept. This prototype will only be used to demonstrate that the chosen architectural style works with a distributed expert knowledge module.

3.3 Research goals and research questions
Aims are: (i) finding an architectural style for distributed ITSs where the architecture consists of a learning environment and an external expert knowledge module. (ii) A proof of concept of a distributed ITS using the chosen architectural style.

To realize the research goals the following research question and sub questions are formulated:

How can we compare and evaluate REST, Web services and Plug-in architectural styles for the distribution of the expert knowledge module in an intelligent tutoring system?
This research question needs to be divided into separate sub research questions. These sub research questions need to be answered before we can formulate an answer to the research question.

1. What is a distributed expert knowledge module and which parts can be distributed?
2. What quality attributes are important for a distributed ITS environment?
3. How do the three architectural styles score on the quality attributes that are important?
4. How do we wrap up the expert knowledge module as a distributed component?
5. Are there frameworks that can be used for building a prototype using the IDEAS framework?

3.4 Research contribution

The contribution of the research can be divided into the following parts:

- A definition of what an expert knowledge module is and what objects it consists of.
- The quality attributes that are important for distributed ITSs.
- Comparison of the three architectural styles against these quality attributes.
- An architectural view for a distributed expert knowledge module in an ITS. This architectural view is based on the architectural style that scores best on the quality attributes.
- The expert knowledge module wrapped up as a distributed object.
- A prototype for a distributed ITS environment.

For this research project, the IDEAS framework will be used for developing the prototype. IDEAS is a framework for developing domain reasoners for ITSs. The language that is used in IDEAS is the functional programming language Haskell [31, 32].

The first step in this research is to find out what an expert knowledge module is, which parts it contains and how it can be distributed.

The second step is to research which quality attributes are important for a non-distributed ITS and an ITS with a distributed expert knowledge module. The quality attributes in the ISO 25010 standard [30] will be used. It is then possible to compare non-distributed ITSs and distributed ITSs. This comparison highlights the difference between non-distributed ITSs and distributed ITSs in terms of software quality. With these results we can identify the quality attributes the architectural style should cover when distributing the expert knowledge module.

The third step is to score the architectural styles from section 2.3 against the quality attributes mentioned in the second step. This way we can find out for every architectural style which quality attributes are covered and which quality attributes are lacking.

The architectural styles can then be compared to the quality attributes important for distributed ITSs to find a suitable architectural style.

The next step will then be to research how the expert knowledge module can be distributed in the designed architecture. For example: in a web service architectural style we need to define which part of an expert knowledge module should be a service (only the domain reasoner or the domain reasoner and the exercise collection). In the case of a REST architectural style we need to find out what we consider as a “resource”.
Finally, a prototype will be developed to demonstrate that the chosen architectural style works with a distributed expert knowledge module. This prototype will be made for an IDEAS tutoring system.

4. Architectural quality attributes for ITSs
To select an architectural style for ITSs with a distributed expert knowledge module, we need to know which quality attributes are important for such an ITS. This is because the software architecture determines whether the system will be able to exhibit its desired quality attributes [1]. If, for example, high performance is important in the system, the architecture must support minimized latency. If security is the most important principle in the system, the architecture must allow secure messaging between objects. This chapter answers research sub question 2: ‘What quality attributes are important for a distributed ITS environment?’. With the answer to research sub question 2, we are able to compare the architectural styles against those quality attributes and identify the trade-offs that come with selecting a particular architectural style for distributed ITSs. In section 4.3 we give an answer to research sub question 1 ‘What is a distributed expert knowledge module and which parts can be distributed?’.

In this chapter the quality attributes of the ISO 25010 quality model [30] are used to define the quality of ITSs. The reason for choosing the ISO 25010 model is that this is an international standard used to describe software quality by giving a taxonomy of quality attributes.

4.1 ISO 25010
The ISO 25010 standard was released in 2011 by a working group of the International Organization for Standardization. It is the successor of the ISO 9126 standard. For more information about ISO 9126 and the reason why the successor ISO 25010 is developed, see Wagner [2]. See Figure 3 for the ISO 25010 product quality characteristics.
The ISO 25010 consists of two models for quality: the product quality model (Figure 3) and a quality in use model. The quality in use model covers the interaction between the system and the different stakeholders. It can be used to further analyze the usability quality attribute [2]. Since the ISO standard does not prescribe when to use which model [30] and since the focus of this research is on finding the quality attributes of ITSs and architectural styles, only the product quality model will be used.

Figure 3. ISO 25010 Product quality characteristics [30]
4.2 Quality for ITSs
To analyze which quality attributes are important for distributed ITSs we first need to define the parts an expert knowledge module consists of. Then we need to determine who the stakeholders are and how they work with ITSs. With this information we can make a qualitative analysis to identify the quality attributes important for non-distributed ITSs. After we identified the quality attributes for non-distributed ITSs, we can make a quantitative analysis on these quality attributes to clarify how they are affected by the distribution of the expert knowledge module. We then can compare distributed ITSs with non-distributed ITSs with respect to software quality. We need this comparison to clarify what the difference is between distributed ITSs and non-distributed ITSs in terms of quality. For example: the advantage of the replaceability quality attribute (for distributed ITSs) might come with a low score on time-behavior. In conclusion: we are using this analysis to identify the quality of distributed ITSs using the quality aspects of non-distributed ITSs.

4.3 Distributed expert knowledge modules
The expert knowledge module of an ITS consists of two separate parts: the collection of exercises and a module that can reason about the problem domain. The latter is also called a domain reasoner [7]. The domain reasoner is responsible for providing stepwise feedback on exercises to students. Heeren and Jeuring consider the expert knowledge module as a module that includes the exercises [7] and in the IDEAS framework the expert knowledge module therefore contains both the domain reasoning and the exercise collection. There are also ITSs that make a distinction between the domain reasoning and the exercise collection. ActiveMath is an example of such an ITS [9]. ActiveMath uses external domain reasoners, but selects the exercises from a knowledge base (see page 103 in Goguadze [9] for an overview of the ActiveMath exercise subsystem). We consider both situations in our research. In our prototype only the situation with an expert knowledge module that includes the exercises is considered.

4.4 Stakeholders and concerns
To make a selection of quality attributes we identify the stakeholders and describe how they work with ITSs. The latter will be described in the form of brief concern descriptions.

The ISO 25010 identifies three types of users: primary users, secondary users and indirect users [30]. In this thesis we will use this ISO 25010 selection of users to categorize typical ITS stakeholders. The stakeholders are:

- Primary users: students
- Secondary users: teachers (for providing content), System administrators and developers
- Indirect user: customer (e.g. universities, commercial educational institutions)

For each stakeholder we now can define concerns. These concerns are defined by what stakeholders need in an ITS to reach their educational goals. This means that there are concerns already fulfilled by ITSs and concerns related to the rationale of this research. An example of the latter is the concern owners of an ITS have: customer wants to provide education to a broad audience of students or learners. Important here is that the concerns should be assessed concerning the distribution of the expert knowledge module. In other words: if a concern is feasible in a certain non-distributed ITS, we want to
know if the distribution of the expert knowledge module of that ITS limits that concern. On the other hand: if non-distributed ITSs score low on a quality attribute important for the rationale of this research, the distribution of the expert knowledge module should increase the score on that quality attribute. This way we want to discover what the trade-offs are for having a distributed expert knowledge module. We then can select an architectural style that minimizes the negative effects of the trade-offs and maximizes the advantages of the distributed ITS. The concerns are defined from research in the literature about ITSs in general [3, 6, 13] and ITSs that are used in practice [7, 13]. In section 4.5 we give an analysis on how the quality attributes satisfy these concerns.

**Student**

1a. Student needs the services provided by the inner and outer loop.

1b. The services provided by the inner and outer loop must be provided in an acceptable time-frame.

**Teachers**

2a. Teachers need to have access to the ITS for developing and adding exercises for the expert knowledge module.

2b. Teachers want to work with the ITS that reacts in an acceptable time-frame.

**System administrators**

3. Administrators connect their learning environment to a certain distributed expert knowledge module.

4. Administrators update the learning environment to new versions.

5. Administrators update expert knowledge modules with new exercises (exercises provided by teachers who do not have the time or have little knowledge about ITSs to add their exercises to that particular ITS themselves).

6. Administrators monitor and recover connections between the learning environment and the expert knowledge module.

**Developers**

7. Developers build and maintain new expert knowledge modules.

8. Developers build and maintain new learning environments.

9. Developers scrutinize and improve existing expert knowledge modules.

10. Developers build connections between learning environments and expert knowledge modules.
Customer

A customer is the party that acquires a certain ITS and this should not be confused with the end user of an ITS.

11. Customer acquires an ITS that meets the educational needs.

12. Customer wants to have a good cost – learning opportunity ratio. E.g., when an ITS is acquired it should support education for a reasonable period of time.

13. Customer wants to provide education to a broad audience of students or learners.

4.5 Quality attributes for distributed ITS environments

In this section a selection of quality attributes from the ISO 25010 will be described. This selection is based on an analysis of which quality attributes contribute to satisfying the stakeholder concerns. Since architectures for non-distributed ITSs already exist, the focus will be on the impact on the quality attributes when distributing the expert knowledge modules. In other words, when we turn a non-distributed ITS into a distributed ITS, which quality attributes will be affected by this transition, positively or negatively. The quality attributes should be measurable.

**Performance efficiency** is important, because *feedback is most effective when occurring in direct response to the need of students* [3]. Ramsay et al. [4] describe that a waiting time exceeding 10 seconds is too long. Users will lose interest. Goguadze mentions in his performance test of the ActiveMath ITS that a 3 seconds response time is still acceptable and that 9.3 seconds is “practically unusable” [9]. Since immediate feedback is one of the design principles of ITSs [6] and that 0.1 second is a response time where users feel that a system is reacting instantaneously [7], we can conclude that a response time of 10 seconds is too long in an ITS. The reaction time in a distributed ITS must therefore be closer to 0.1 second than to 10 seconds.

Since a distributed system will always be slower than the same non-distributed system, we want to minimize latency. Thus especially the *time behavior* aspect is important since students and teachers want ITSs that have an acceptable response time (concerns 1b and 2b). Also *capacity* is important since ITSs need to handle many users simultaneously [7].

**Maintainability** is an important aspect for tutoring systems. Especially *reusability, modularity* and *analyzability* are important sub characteristics of maintainability. Reusability is important because it increases the efficiency of building ITSs [5]. One of the reasons to distribute expert knowledge modules is to give ITS developers the possibility to use already existing expert modules in their ITS. Modularity is important to make it possible to distribute the expert knowledge module to be used by external ITSs. This attribute is especially important for developers since they need to develop and scrutinize external expert knowledge modules (concerns 7 to 10). An ITS scores well if developers are able to reuse expert modules in their own ITSs with an amount of effort that is less than if they had developed an ITS from scratch.

**Compatibility** is a quality attribute needed to make *interoperability* possible between the learning environment and external expert knowledge modules. The learning environment must be able to
communicate with distributed expert knowledge modules. Concerns 6 and 10 are affected by how this quality attribute scores in an ITS.

For the **Security** attribute especially the **integrity** and **confidentiality** attributes may become important when the expert knowledge module is distributed over internet. Since the internet is a public network, there is an additional risk of data leakage between components. An ITS scores well on the security attributes if data stored in an ITS is protected against loss and unauthorized access.

With the distribution of the expert knowledge module there will be a layer between the learning environment and the expert knowledge module in the form of an interface. This means that there is an additional point of failure concerning the **availability** of ITS functionality. Therefore, the quality attribute **Reliability** will also be considered. Reliability depends on many factors such as server uptime, load balancer performance during peak time and requests from teachers for reports with large amounts of data [8]. Availability is called ‘robustness” in Patvarczki et al. [8]. In our research the reliability of an ITS is considered as an characteristic of that ITS. When the expert knowledge module is distributed (from that particular existing ITS) there is a risk that it affects reliability negatively. To measure this quality attribute: a distributed ITS should still be able to fulfill concerns 1a and 2a.

In the comparison between non-distributed ITSs and distributed ITSs **Portability** needs to be considered, since the developers of distributed ITSs must be able to efficiently build an interface between the learning environment and the expert knowledge module (concern 10). Building such an interface efficiently means that the building effort is less than building an ITS from scratch. An efficient way of building these interfaces is, for instance, using existing frameworks such as Servant [10], which is a framework for building Web API’s in Haskell. A special note must be made on the sub characteristic **adaptability**. A distributed ITS scores well on this sub characteristic because the distribution of expert knowledge modules makes them available to multiple ITSs. This is important for fulfilling concern 13 (Customer wants to provide education to a broad audience of students or learners). Concern 13 is important because there is a coherency with this concern and the research rationale to increase access to education provided by ITSs.

Teachers must also be able to efficiently transfer their learning content from one expert knowledge module to the other (concern 2a and 2b). Also system administrators and developers need a system that scores well on especially **replaceability and installability** (concerns 3 to 6 and concern 10). An ITS scores well on those sub characteristics if the workload for administrators, developers and teachers is the same as it is for equivalent learning systems such as CAI’s and electronic learning environments.

**Functional suitability** is a quality attribute that is of interest for students using ITSs (concern 1a) and customers (concern 11). This quality attribute will be measured for the difference between non-distributed ITSs and distributed ITSs only. In other words, we are only interested in this quality attribute if the distribution of the expert knowledge module will affect it. E.g., if an ITS has a good score on functional suitability, the distribution of the expert knowledge module should not affect this score in such a negative way that reluctance of students and customers to work with that ITS becomes a risk.
Table 1. Comparison between non-distributed and distributed ITSs

<table>
<thead>
<tr>
<th>Quality Characteristic</th>
<th>Non-distributed ITS</th>
<th>Distributed ITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional suitability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability - Availability</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>- Recoverability</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>Performance efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Time behavior</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>- Capacity</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>Usability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintainability - Reusability</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>- Modularity</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>- Analyzability</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>Security - Integrity</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>- Confidentiality</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>Compatibility - Interoperability</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>Portability - Replaceability</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>- Installability</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>- Adaptability</td>
<td>⚫</td>
<td>⚫</td>
</tr>
</tbody>
</table>

Good
Sufficient
Bad

In Table 1 non-distributed and distributed ITSs are compared using the eight quality characteristics derived from the ISO 25010 standard. Sub quality characteristics are only taken into account if they are affected by the distribution of the expert knowledge module or if they are affected by the choice of architectural style. Therefore, functional suitability and usability are not split into sub characteristics and are scored as sufficient in both systems since these characteristics indicate if a system has functions that meet stated and implied needs (functional suitability) and how users perceive a software system (usability) [30]. When distributing the expert knowledge module, we do not change the functions and usability aspects of ITSs. The comparison in Table 1 is made on an abstract level and it is not a comparison between existing systems. The papers of Murray [5], Corbett et al. [6], Heeren and Jeuring [7], Patvarczki et al. [8] and Goguadze [9] were used for the scoring of non-distributed ITSs.

To give an example of how to interpret Table 1 we highlight the security characteristic. The sub characteristics of security that are affected are integrity and confidentiality. Both score sufficient in a non-distributed ITS, because data about students is protected against loss and unauthorized access by the fact that in a non-distributed ITS user access comes from a private network (the school’s network) and data between the ITS components resides on that private network. Besides the existing security
aspects for non-distributed ITSs, system hacking needs to be considered in distributed ITSs as well, since the expert knowledge module resides on the internet, which is a public network. This means that if the expert knowledge module of that particular non-distributed ITS is distributed, this ITS will score badly on security if no countermeasures are taken. In general, Table 1 tells us that if the score of a quality attribute for an ITS after distributing the expert knowledge module is lower than a non-distributed ITS, it becomes an attention point in the selection of an architectural style.

In Table 1 we see that there are a few quality attributes that need more attention than those in a non-distributed ITS. Reliability, performance efficiency and security score from good to sufficient in non-distributed ITSs. Since the distribution of the expert knowledge module has an impact on those quality attributes, these quality attributes will score badly (compared to the non-distributed ITS) if we simply distribute the expert knowledge module without using an architectural style. Note that the ‘good’ score of performance efficiency depends on a maximum simultaneous use of 150 users [9].

For the scoring of the maintainability characteristic the information from Heeren and Jeuring [7] and Nwana [3] are used. They describe a non-distributed ITS as a cooperation of modules with their own responsibilities. This means a good score on modularity and a sufficient score on analyzability, but a low score on reusability. The latter because it does not provide an interface for interaction between modules outside the ITS. After the distribution of the expert knowledge module the sub characteristics reusability, modularity and analyzability will score well because these sub characteristics are inherent to distributed systems. Murray [5] states that these sub characteristics benefit of a decentralized architecture.

In the literature we have not found information about usability and portability for non-distributed ITSs. Therefore, these quality attributes are scored as ‘sufficient’ (for non-distributed ITSs).

In Heeren and Jeuring [7] interoperability (a sub characteristic of compatibility) is considered important for distributed feedback services. Interoperability is the “degree to which two or more systems, products or components can exchange information and use the information that has been exchanged” [30]. A non-distributed ITS scores badly on this characteristic, because it is not able to use external components. The existing ITSs that use distributed expert knowledge modules without a standard interface score sufficiently on interoperability. It is not the ideal situation, but it works sufficiently for exchanging data between the learning environment and the distributed expert knowledge module [7, 9, 32].

4.6 Evolution of ITSs
To support concern 12 (“Customer wants to have a good cost – learning opportunity ratio.”) the evolution of ITSs must be taken into consideration. When an ITS is acquired it cannot be the case that after, for example, three years the system is outdated and has to be replaced by a new system. The main reasons that software becomes outdated is the incorporation of new user requirements [11] and changes in the software environment [11, 12]. One of the aspects to cope with new user requirements and changes in the software environment is the software architecture. It should allow “considerable unanticipated change in the software without compromising system integrity” [11]. When selecting the architectural style and designing an architecture for distributed ITSs it is important to take software evolution into consideration. In other words, in the ITS architecture it must be possible to change a component without affecting other components. It is important to realize that software evolution is not a sub characteristic of maintainability. Software evolution is a particular phase in the software lifecycle.
where changes in the software are implemented to meet major (unpredicted) changes in user requirements [11].

5. Scoring architectural styles on the quality attributes important for distributed ITSs

In this chapter the following three architectural styles will be considered:

- REST architecture
- Service oriented architecture with web services
- Plug-in architecture

Each architectural style will be briefly described and scored with a weighting score for the quality attributes described in Table 1. This quantitative analysis answers research sub question 3: ‘How do the three architectural styles score on the quality attributes that are important?’

5.1 REST architectural style

Fielding and Taylor describe REST as an architectural style that focuses on three goals: minimizing latency, maximizing scalability and independence of components [19]. The REST architectural style is based on the Hypertext Transfer Protocol (http).

The REST architectural style consists of three classes of architectural elements:

- Data elements
- Connectors
- Components (processing elements)

The data elements are a key aspect of REST. Data elements are resources that can be used by human readers (e.g. an article or an image) and machines (mobile code such as JavaScript). Those data elements are identified by URLs (Uniform Resource Identifiers). A client (human or machine) requests a data element by calling its URI. A representation of that data is then sent to the client. A main point of REST is thus that the client is not moved to the data, but the data is moved to the client. Another important concept is that not the data itself is sent, but a representation of that data and the meta-data about how to render this representation (e.g., HTML).

The data elements can be manipulated with methods. In REST, the same set of methods is used for all data elements. This in contrast with, for example, web services where each service has its own set of specialized methods (see Figure 4). To give an example: in Figure 4 we see that for each data element the same methods are used: GET, PUT, POST and DELETE. These four methods are used for retrieving and manipulating data elements.

The connectors in REST present an abstract interface for communication [19]. Important is that all REST connectors are stateless. Therefore each, request must contain all of the information necessary for a connector to understand the request. The advantage is that a server does not need to maintain session information about the communication with a certain client. This improves the scalability characteristic, because a server can be replicated without concerning stored session information (since there is none).
REST components are the processing elements in a REST architecture. There are four components in REST [19]: origin server, gateway, proxy and user agent.
A user agent is a client that initiates a request and is the recipient of the response (a representation of the requested resource). An origin server is the location of resources. Proxies and gateways are intermediaries between clients and servers. The difference between those two is that a client determines which proxy to use (if any) and a gateway is set up by the administrator of a server.

5.2 REST quality aspects

REST focuses on the three goals: minimizing latency, maximizing scalability and independence of components. Scalability influences the performance efficiency quality attribute, because scalability enables a system to adapt to change in the number of active users. Capacity as the sub characteristic of performance efficiency is also influenced by the availability of system resources (to active users). We therefore interchange scalability for the ISO 25010 quality sub characteristic ‘capacity’. In this section we analyze what the three REST goals tell about the quality of REST using the ISO 25010 standard. In Fielding [20] the REST architectural style is defined by six constraints: client-server, stateless server, caching, uniform interface, layered system and code on demand (optional). For detailed information about these constraints see Fielding [20]. These constraints define the quality of the REST architectural style.

The sub characteristics replaceability and installability of the portability quality attribute and the capacity quality are covered by the client-server constraint. Reliability is covered by the stateless constraint because the stateless constraint eases the task of recovering from partial failure. The stateless constraint brings the disadvantage of decreasing network performance since the repetitive data is sent in a series of requests [20]. To solve this problem the caching constraint is introduced in REST to improve the network performance. Unfortunately it comes with the trade-off that it can decrease functional suitability (in this case the sub characteristic ‘functional correctness’) because data from a cache can be stale compared to the same data that would have been obtained if the request was sent to the server directly. Finally, the layered system allows to implement security policies.

The requester of a data item is not moved to the data, but a representation of the data is sent to the requester. This way of data processing gives the advantage of encapsulation and evolution of services [20].
In Table 2 the scoring of the REST architectural style is depicted using the analysis of REST and the sub quality attributes from Table 1. The quality attributes performance efficiency, portability and reliability are covered in REST. Time behavior (a sub characteristic of ‘performance efficiency’) is covered by the use of the caching constraint. As mentioned before, the caching constraint improves the network performance. Adaptability as the sub characteristic of portability is covered because REST is designed to give access to resources on as many platforms as possible [20]. Reliability is covered by the stateless constraint. The sub characteristic recoverability is covered by REST because the stateless constraint enables recovering from partial failures [20] and availability is covered because research showed that web protocols that uses REST as a design model have had a positive effect on the robustness of those protocols [20]. Note that availability is sometimes referred to as robustness (see section 4.5).

Reusability and analyzability as the sub characteristics of maintainability are covered by the fact that it is easy to develop new data elements. This is because all the data elements can be manipulated by the same set of methods instead of developing new methods for new data elements (reusability). Since data elements are a key aspect of REST it is relatively easy to scrutinize modules by analyzing the data they consist of (provided that the data is open source). For example, in a distributed ITS the expert knowledge module is a collection of data elements in REST. This collection can be scrutinized by researchers on correctness and completeness. The sub characteristics modularity is not covered by the REST architectural style itself. The coverage of this sub characteristics depends on how the software system is decomposed into modules.
The quality attribute security is partly covered. Security is covered by the fact that the layered system constraint enables the installation of security policies [20]. The REST architectural styles does not enforce such a policy however. Compatibility is also partly covered in REST. Any system that wishes to use the data elements need to use the REST way of communicating to those data elements. In other words, such system cannot expect the connectors to the data elements to remember the session, since communication needs to be stateless. The ability for a system to be able to send all the necessary information for the connector to understand the request strongly depends on the statelessness of that system.

Functional suitability is a quality attribute that is not covered by using the REST architectural style. With using only a limited set of methods it is arguable that, for instance, the functional completeness sub quality attribute decreases. To the best of our knowledge, there is no information about how REST affect the usability quality characteristic. We therefore consider usability also as not covered in REST.

5.3 Service oriented architecture with web services

The service oriented architectural style (SOA) focuses on the composition of services across disparate pieces of software [21]. A service requester initiates the execution of a service by sending a message to a service provider. The service provider executes the service with the message as input and provides the results to the service requester. In an ITS the learning environment can be considered as the requester and (a part of) the expert knowledge module can be considered as the service provider. It is important to realize that the difference between a service and a software functionality is that the first is defined at a higher level of abstraction. Where a functionality is an output from a software object (return value); a service is an integration of functionalities.

A service provider can also be a service requester to other services. To provide a particular service to a service requester, a service provider may need services from other service providers.

Advantages of SOA are [21]:

- **Reuse** – The ability to create services that are reusable in multiple applications.
- **Efficiency** – The ability to quickly and easily create new services and new applications using a combination of new and old services.
- **Loose technology coupling** – The ability to model services independently of their execution environment and create messages that can be sent to any service.

For ITSs there is the extra advantage that it is possible to encapsulate educational content inside a (web) service to increase interoperability and reusability [17].

For a service provider to provide services to a service requester the communication exists of two parts:

- A service description: this is a public interface describing how to find and use the service. There are two common web service standards for this: Web Service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI) [16].
- A message exchange protocol for the communication between the service provider and service requester.
A service oriented architecture is not a technique, but a paradigm that can be implemented by a suitable technology platform. The platform described in this research is the web services technology. A web service is a technology for implementing a service oriented architecture. The key idea is that web services use the internet as a communication infrastructure. To do so, there is a set of (open) standards for communication, coordination and orchestration of services. For more details on web services in general, see Newcomer [21].

If we want to use the Service Oriented Architectural style using web services for the learning environment, we need to wrap the expert knowledge module in a service or a set of services that can be provided to learning environments via the internet.

Figure 5 gives an example of expert knowledge modules in a web service architecture. The expert knowledge modules are wrapped as web services and are thus external to the learning environment. The web services are registered to an UDDI registry. Learning environments from different universities and commercial third parties can use the expert knowledge module they need as a service. The learning environment uses the UDDI to look up an appropriate service and then uses the WSDL to communicate with the public interface of that particular service. The learning environment invokes the web service to get the desired feedback service.

To use the web service architectural style for the connection between learning environments and expert knowledge module, we need to use a Web Service Description Language to describe the public interface
of the expert knowledge modules; the learning environment needs to know how to use the service to send XML-messages that the service understands. The learning environment also needs to know where to find the expert knowledge modules it needs. This needs to be done by consulting a registry with published web services. An example of such registry is the UDDI registry [16]. If the service oriented architectural style is used for the distributed expert knowledge modules, we need to investigate how to implement the WSDL, Web Service interfaces and message exchange protocols and how to design an UDDI. This immediately shows the disadvantage of Web Services: they have large and complex protocols.

Another implementation of SOA for e-learning systems is the service oriented cloud computing architecture (SOCCA) [22]. In this theoretical implementation of SOA the services are a layer in the cloud computer layered model, where the demand layer with e-learning portal web server and the UDDI registry communicates with the services encapsulated into the delivery layer [22]. SOCCA has the benefits of improving scalability (capacity in the ISO 25010 standard), modularity and security. It also has the disadvantage of the so called data lock-in. Data lock-in is the problem that arises when transferring data from one cloud-computing vendor to another vendor. Each vendor uses his own storage APIs that have not been subject to active standardization [24]. Data lock-in decreases the sub characteristics analyzability and interoperability. SOCCA also is a theoretical and not yet an implemented architecture. Therefore, in our research we will only consider the web services implementation of SOA, since this is a proven technology. We consider the research on SOCCA as a possible architectural style for distributed ITSs as future work.

5.4 Quality aspects of the Service Oriented Architectural style
The SOA architectural style focuses on reuse, efficiency and loose technology coupling. The focus on reuse means that systems built with the service oriented architectural style score well on the maintainability aspect of the ISO 25010 model. SOA works with a service requester requesting a service from a service provider by sending messages. This means that a service can be used by any client (another service or user) that knows how to query the public interface. The details about this interface can be found in the registry. We therefore argue that SOA scores well on compatibility. There is also a downside. Because the interaction between services is done by messaging there is the risk of messages being intercepted by hackers and the risk of lost messages. The messaging system is also slower than direct interaction between components. This means that the advantages of SOA come with the price of a low score on security, time behavior and reliability.
Table 3. SOA architectural style

<table>
<thead>
<tr>
<th>Functional suitability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
</tr>
</tbody>
</table>
| - Availability         | [ ]  
| - Recoverability       | [ ]  
| Performance efficiency |  
| - Time Behavior        | [ ]  
| - Capacity             | [ ]  
| Usability              | [ ]  
| Maintainability        |  
| - Reusability          | [ ]  
| - Modularity           | [ ]  
| - Analyzability        | [ ]  
| Security               |  
| - Integrity            | [ ]  
| - Confidentiality      | [ ]  
| Compatibility          |  
| - Interoperability     | [ ]  
| Portability            |  
| - Replaceability       | [ ]  
| - Installability       | [ ]  
| - Adaptability         | [ ]  
|  
- Covered
- Partly covered
- Not covered

In Table 3 the scoring of the SOA architectural style is depicted using the analysis of SOA implemented by web services and the sub quality attributes from Table 1. We see that adaptability, reusability, modularity, interoperability and replaceability are covered by SOA. Adaptability is covered because web services provide multiple channels of access to their services [21]. Interoperability and reusability are increased by encapsulation of educational content inside web services [17]. The modularity sub characteristic is covered by SOA because in SOA software functionalities are wrapped as services with their own responsibilities that can use other services if necessary. Because SOA focuses on loose technology coupling, the replaceability sub characteristic is covered. O’Brien et al. [23] mention that changes in functionality that do not change the web service interface are relatively easy to implement and for changes that do change the web service interface the interface changes need to be registered with the UDDI. Replaceability in SOA is therefore covered because of this possibility to replace services.

The security quality attribute is partly covered because SOA works with messages between users and services and between services mutually. As mentioned before, this increases the risk of lost messages and hacking. In the web services implementation of SOA a specification framework for security has been developed called WS-Security [21] for secured messaging. This technology standard is continuously updated [22]. Therefore, the security attribute is scored as ‘partly covered’ in Table 3.

Availability, capacity and recoverability are also partly covered. O’Brien et al. [23] mention that availability can technically be covered by techniques such as exception handling. When an invoked service is not available, another source for the needed service can dynamically be located. Capacity can be covered by deploying the same service to multiple platforms. Intermediaries for data marshalling are
needed to increase capacity and that comes with the price of a decreased time behavior. Concerning recoverability, Newcomer [21] mentions recovery as an aspect of reliable messaging. The WS-reliability specification is available in the web services standard to enable recovery to a known state.

Functional suitability, usability, time behavior, analyzability and installability are not covered by SOA. We have not found information about the impact on functional suitability and usability by using SOA. As mentioned earlier in this section, SOA scores low on time behavior because of the use of messages instead of method invocation. O’Brien et al. [23] also mention the overhead of looking up services in a directory as a negative impact on time behavior. Because functionality is wrapped up as a service and is behind an interface, SOA does not cover analyzability. SOA scores low on installability since developers need to develop and work with large and complex protocols.

5.5 Plug-in architecture
A plug-in architecture is designed to allow applications to be extended by plug-ins (or add-ons). The plug-ins are software components that can add new behavior to applications. For a plug-in to be able to extend an application, it needs to use the provided interface of the application. This means that it is also possible for third parties to design new plug-ins for an application. Those third parties can do that by implementing the interface of that application in their plug-ins. The main idea of the plug-in architectural style is that the application invokes the plug-ins and not the other way around. The advantage is that this architectural style enables variability and late binding of software components. Examples of systems that use the plug-in style are Eclipse, Firefox and Adobe Photoshop.

The plug-in architectural style is used in the tutoring system CALAT [18, 25] for animated simulation. The advantage is that because of the plug-in architecture it is easy to update this system with more sophisticated simulations when the technique becomes available.

In the architecture for the distributed expert knowledge modules we can see the learning environment as the host application that can be extended by an expert knowledge module as a plug-in (see Figure 6).
The ITS architecture in Figure 6 consists of the four modules mentioned in Figure 1. The Expert knowledge module is a placeholder for plug-ins. If the plug-in architectural style will be used, it needs to be investigated how to build the expert knowledge module as a plug-in and how to enable the learning environment to use these plug-in expert knowledge modules. Because plug-ins are not compiled into the code of the host application (they are linked with the host application via interfaces [26]), we need to find out how to design the interface between the host application and the plug-ins. Since a domain reasoner reasons about exercises in an exercise collection, we also need to investigate if the exercise collection and the domain reasoner should be one plug-in or that they should be separate plug-ins. In the latter situation we also need to find out how the two plug-ins should interact (since a logic domain reasoner cannot reason about math exercises). See Figure 6 for an example of both situations.

The advantages of the plug-in style for distributed ITSs are: the possibility to use different expert knowledge modules for different learning environments, the possibility to choose whether or not to use a certain version of an expert knowledge module and, once an expert knowledge module is downloaded from the distributed plug-in repository and installed as a plug-in into the learning environment, the ITS can be used offline.

There are also a few disadvantages with the plug-in architectural style considering ITSs [14]:

- Plug-ins are platform specific
- Code needs to be distributed to the user (in this case the learning environment)
- Code needs to be installed on the client machines (running the learning environment)
- The distribution and installation problem recurs with every update of an already installed plug-in

Sherman et al. [14] propose the plug-in architectural style as an alternative architecture for a distributed user interface. To the best of our knowledge, a situation with a distributed expert knowledge module in a plug-in architecture has not been investigated yet.

5.6 Quality aspects of the plug-in architectural style

In Table 4 the scoring of the plug-in architectural style is depicted using the analysis of the plug-in architectural style and the sub quality attributes from Table 1.
In Table 4 we see that functional suitability, availability (sub characteristic of reliability), performance efficiency, maintainability and adaptability are covered in the plug-in style. Functional suitability is covered because the host application can be “extended with plug-in components tailored to the users' needs” [27] and thereby gives flexibility in functionality.

Availability is covered by the fact that a software system in the plug-in architectural style can be used offline. Even when the plug-in repository is not available, it is still possible to use the software system with the already downloaded and installed plug-ins.

Since a system with a plug-in architecture can be used offline it will score better on the time behavior characteristic than a system that relies on a module that remains on a distributed server. Also the fact that installed plug-ins are only invoked when needed increases performance and thus improving time behavior [26]. Because of the offline availability of downloaded plug-ins it is possible to scale up by using multiple installations of a system with its own set of installed plug-ins. This local scaling up is a great advantage compared to systems where a functionality is used in a distributed fashion.

Maintainability is covered because plug-ins can be built by third parties as long as they implement the host application’s interface. This gives a good score on reusability because the plug-in can be used in other host applications (after changing the plug-ins interface to the host application’s interface).

Modularity is covered by the separation of responsibilities. For example, a distributed ITS does not need to maintain a complete model of the student actions because that is the responsibility of the host
application [28]. When third-party open source plug-ins are provided from a public repository they are accessible for analyzability.

Adaptability as the sub characteristic of portability is covered because the plug-in architectural style gives a large freedom to the owner of the system to adjust the system to the user’s needs. This freedom does not only concern functionality, but also the platform the system will be used on. By refactoring plug-ins into smaller plug-ins it is possible to use the same system on a device with limited resources such as smart phones [26].

We have not found information about Recoverability and replaceability. Because of the many possible functionality configurations (the set of plug-ins used in a certain program implementation), it can be difficult to recover or replace that certain implementation. Therefore we can assume that the plug-in style only partly covers these quality attributes.

Security, compatibility, installability and usability are not covered in the plug-in architectural style. Security is an important concern. The plug-in architectural style enables arbitrary plug-ins to be installed (from the internet, for example) and they are allowed to have access to the system they plug into. The risk is that it is possible to install plug-ins that contain serious bugs or malicious code [26]. This security problem can be mitigated by only allowing plug-ins that come from a trusted party. Another security measure is using plug-ins that are checked on the absence of malicious code by trusted third parties. However, these security measures still rely on trusting the download site or the trusted party. A more technical solution is running plug-ins only in a secure environment, a so-called sandbox [27]. Because of the high risk on buggy or malicious code and the limited countermeasures, security scores low in the plug-in style.

Plug-in version management is an issue because the plug-in style enables to install different versions of the same plug-in next to each other. This concurrent plug-in versions is a problem because plug-ins may require functionality from other plug-ins and very often there are strict criteria about what versions are required [26]. This can cause the situation that plug-ins cannot collaborate with each other. This results in a low score on compatibility. The plug-in style has the disadvantage that plug-ins are platform specific and that code needs to be distributed to and installed on the client site. These problems occur with every plug-in update. This gives a low score on installability. Since we have not found information about the impact on usability by using the plug-in architectural style, we consider this quality attribute as not covered either.

Ritter [28] gives a description about a learning system in the plug-in style where the learning system itself (called ‘tutor’) is used as a plug-in in an existing system (the tool). Ritter mentions that one reason to maintain the distinction between tool and tutor has been to address technical migration of the software. Ritter also gives some examples of the tutor being used in newer versions of the tool. This gives a good example why the plug-in style scores well on software evolution. Every part of the system (plug-ins) can evolve independently.

6. Comparison of architectural styles and distributed ITSs
Using the scoring of distributed ITSs from chapter 4 and the scoring of the architectural styles from chapter 5 we can analyze how the architectural styles match the quality attributes of distributed ITSs. In
In this chapter, we found the quality attributes important for ITSs. In Table 1 the quality attributes that are scored from sufficient to bad for distributed ITSs are the quality attributes an architectural style should cover or partly cover in a distributed ITS. This is important, because if a quality attribute scores badly in the generic distributed ITS from Table 1 and it is also not covered in the architectural style the concerns that depend on that quality attribute are not met.

A ‘perfect match’ would be one where the architectural style for distributed ITSs covers all the quality attributes that scored ‘sufficient’ to ‘bad’ in Table 1. If in our quantitative analysis in this chapter none of the architectural styles is a ‘perfect match’ then the selection of the architectural style must involve trade-offs between quality attributes.

### Table 5. Scoring of architectural styles

<table>
<thead>
<tr>
<th></th>
<th>Distributed ITS</th>
<th>REST</th>
<th>SOA</th>
<th>Plug-in</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functional suitability</strong></td>
<td>○ 0.5</td>
<td>○ 0.0</td>
<td>○ 0.0</td>
<td>● 1.0</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Availability</td>
<td>○ 0.0</td>
<td>● 1.0</td>
<td>○ 0.5</td>
<td>● 1.0</td>
</tr>
<tr>
<td>- Recovery</td>
<td>○ 0.0</td>
<td>● 1.0</td>
<td>○ 0.5</td>
<td>○ 0.5</td>
</tr>
<tr>
<td><strong>Performance efficiency</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Time behavior</td>
<td>○ 0.0</td>
<td>● 1.0</td>
<td>○ 0.0</td>
<td>● 1.0</td>
</tr>
<tr>
<td>- Capacity</td>
<td>● 0.5</td>
<td>● 1.0</td>
<td>○ 0.5</td>
<td>● 1.0</td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td>● 0.5</td>
<td>● 0.0</td>
<td>○ 0.0</td>
<td>○ 0.0</td>
</tr>
<tr>
<td><strong>Maintainability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Reusability</td>
<td>● 1.0</td>
<td>● 1.0</td>
<td>● 1.0</td>
<td>● 1.0</td>
</tr>
<tr>
<td>- Modularity</td>
<td>● 1.0</td>
<td>● 0.0</td>
<td>○ 1.0</td>
<td>● 1.0</td>
</tr>
<tr>
<td>- Analyzability</td>
<td>● 1.0</td>
<td>● 0.0</td>
<td>○ 0.0</td>
<td>○ 0.0</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Integrity</td>
<td>○ 0.0</td>
<td>● 0.5</td>
<td>○ 0.5</td>
<td>○ 0.5</td>
</tr>
<tr>
<td>- Confidentiality</td>
<td>○ 0.0</td>
<td>● 0.5</td>
<td>○ 0.5</td>
<td>○ 0.5</td>
</tr>
<tr>
<td><strong>Compatibility</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Interoperability</td>
<td>○ 0.5</td>
<td>● 0.5</td>
<td>● 1.0</td>
<td>○ 0.0</td>
</tr>
<tr>
<td><strong>Portability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Replaceability</td>
<td>● 0.5</td>
<td>● 1.0</td>
<td>● 1.0</td>
<td>● 0.5</td>
</tr>
<tr>
<td>- Installability</td>
<td>○ 0.5</td>
<td>● 1.0</td>
<td>○ 0.0</td>
<td>○ 0.0</td>
</tr>
<tr>
<td>- Adaptability</td>
<td>● 1.0</td>
<td>● 1.0</td>
<td>● 1.0</td>
<td>● 1.0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>7.0</td>
<td>10.5</td>
<td>7.5</td>
<td>9.0</td>
</tr>
</tbody>
</table>

- Good / Covered
- Sufficient / Partly covered
- Bad / Not covered
Table 5 merges the results from the distributed ITS analysis from chapter 4 and the results of the architectural style analysis from chapter 5. This table enables us to compare the architectural styles concerning their coverage of quality attributes against the quality attributes important for distributed ITSs. In Table 1 we used the scoring of the quality attributes of non-distributed ITSs to analyze how the quality attributes are affected when we turn a non-distributed ITS into a distributed ITS. When comparing architectural styles we are only interested in the scoring of distributed ITSs. Therefore, Table 5 does not contain the non-distributed ITS scoring. In the next section each architectural style will be compared against the quality attributes important for distributed ITSs. For each architectural style we need to answer the following question:

*How well does an architectural style cover the quality attributes important for distributed ITSs?*

With the answers to that question we can highlight the benefits and trade-offs for each architectural style. An informed decision about which architectural style to use is then possible.

Table 5 contains the scoring per sub quality characteristic for each architectural style. The scoring of architectural styles to answer the architectural-style-question is done using the following scoring system:

- One point for each quality attribute that is covered in the architectural style.
- Half a point for each quality attribute partly covered in the architectural style.
- No points for quality attributes not covered in the architectural style.

We use the same scoring for the generic distributed ITS in column one:

- One point for each quality attribute scoring well.
- Half a point for each quality attribute scoring sufficiently.
- No points for each quality attribute scoring badly.

We can use table 5 for comparing the total score of each architectural style with the ITS score in the first column. If a certain architectural style scores higher than the 7 points of the ITS score, we can say this architectural style has the potential of improving the quality of distributed ITSs. We use the word ‘potential’ because we also have to consider the trade-offs. Using a particular architectural style means that some quality attributes will be covered, while other quality attributes will only be partly covered or not covered at all. In other words, the total score in Table 5 for each architectural style answers the question if an architectural style improves the distributed ITS quality. The scoring of the individual quality attributes must also be taken into account to find the trade-offs for the choice which architectural style to use. The trade-offs need to be considered on how the distributed ITS will be used. In the next section we give a situation where the plug-in style is the best choice despite the second best score.

### 6.1 Comparison of architectural styles

Regarding the scoring of architectural styles in Table 5 we see that all three architectural styles score higher than the generic distributed ITS. This means that using any of the three styles will improve the quality of distributed ITSs. The REST architectural style has the highest score (10.5 points).

We also have to take software evolution into consideration when selecting an architectural style. Software evolution is important for supporting the customer’s concern to have a good cost–learning
opportunity ratio. REST supports software evolution. Fielding [20] explains that because REST works with data representations that are sent to requesters, it has the advantage of enabling evolution of services.

The next step is to consider how a particular distributed ITS will be used. In other words, we need to consider the trade-offs for each architectural style. In the description of architectural styles in chapter 5, we see that there is one main difference between REST and SOA on the one hand, and the plug-in style on the other hand. With the plug-in architectural style the plug-in repository is the distributed part and not the use of the software program built on this architectural style. Thus the plug-in style is the only style from the set of architectural styles where the distributed ITS can be used offline. If the educational needs of the customer require an ITS being used offline, the plug-in style could be the appropriate style for such ITS despite the lower score. Because Table 5 also helps us to assess the individual quality attributes, we see that the offline-use-advantage of the plug-in style comes with a trade-off on the security quality attribute. Distributed ITSs already score badly on security and it is not covered in the plug-in style either. Because of the higher score for REST we will use this architectural style to build the prototype. A trade-off we made with this choice is the ‘partly covered’ score on ‘interoperability’. The SOA style scores better (‘covered’) on this quality aspect. However, as mentioned in section 5.2, the negative influence of this quality aspect strongly depends on how stateless the expert knowledge module is. Since we are using expert knowledge modules based on IDEAS we do not expect interoperability to be a problem since IDEAS is fully stateless [32].

6.2 The expert knowledge module as a distributed component
Before we can start with building the prototype, we first need to answer research sub question 4: ‘How do we wrap up the expert knowledge module as a distributed component?’. The answer to this question depends on which architectural style to choose. Distributed expert knowledge modules are wrapped up differently in, for example, the plug-in architectural style than in the REST architectural style. In the REST style we need to provide the expert knowledge as a set of resources. Since we are using expert knowledge modules built with the IDEAS framework, we will consider the expert knowledge module as a module that includes the exercises (see section 4.3). In the next chapter and in our prototype there are resources concerning the strategies and rules of the expert knowledge module as well as resources representing the exercises.

7. Architectural views for distributed ITs
To visualize the architecture of distributed ITs we use two views. Since the prototype is built on the IDEAS framework, the views in this chapter capture the architecture of a distributed ITS built with the IDEAS framework.

To address the ITS stakeholders we need a set of different views of the distributed ITS. Each of the views characterizes the aspects of the distributed ITS important for each group of stakeholders mentioned in section 4.4. Table 6 shows the views relevant for each stakeholder.
Table 6. Relevant architectural views for stakeholders

<table>
<thead>
<tr>
<th>Architectural view</th>
<th>Stakeholder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical view</td>
<td>Students, teachers, customers</td>
</tr>
<tr>
<td>Resource view</td>
<td>Developers</td>
</tr>
<tr>
<td>System view</td>
<td>System administrators</td>
</tr>
</tbody>
</table>

The logical view gives a high level view on how the system works. The resource view shows the resources and the connection between resources.

The system view, important for system administrators, shows how the components of distributed ITSs are portrayed on hardware artefacts. Since this research does not cover the hardware needs of ITSs, we are not able to construct a system view. Therefore we consider the construction of the system view as future work.
7.1 Logical view

Figure 7 gives a logical view of how a distributed ITS using the REST architectural style works. This figure visualizes the feedback services from distributed expert knowledge modules that can be used by a learning environment. In this view we see that each learning environment communicates its requests to a REST interface. Each expert knowledge module uses its own instance of the REST interface. The learning environment sends the URI of the desired resource to the REST interface. The REST interface returns a representation of that resource in the form of HTML, JSON or OpenMath. The latter is a standard for representing the semantics of mathematical objects [34]. The difference between Figure 2 and Figure 7 is that the latter uses the REST architectural style to form a layer between the learning environments and the distributed expert knowledge module. Because this layer provides a standardization in communication between distributed expert knowledge modules and learning environments, we have the reuse and operability advantage mentioned in our research incentive.
7.2 Resource view

Figure 8 gives the resource view of the prototype we designed. We see that the REST layer forms an interface between learning environments and the resources from the expert knowledge module. This interface defines a set of methods, resources, representations and URLs of resources. Each resource has its own unique URI as identifier. A learning environment that uses an expert knowledge module...
implementing the REST layer can use the ‘get’ method in combination with the URI to obtain a resource. For example:

request-URL:  www.expertknowledgemodule.nl/eval.full/examples
request method:  GET

gives a representation of the set of examples from the exercise eval.full from the expert knowledge module implementing the REST layer. A representation can be in formats such as JSON, HTML and OpenMath. See the next chapter about the prototype for the technical implementation.

In the upper right corner of Figure 8 we see the distributed expert knowledge module. To create the resources, the REST layer uses the feedback service from this distributed expert knowledge module.

The information flow between the REST interface and the resources is depicted by arrows specified with cardinality. The arrows describe the dataflow for learning environments to go from one resource to the other. This is called HATEOAS: hypermedia as the engine of application state [33]. With HATEOAS it is possible for learning environments to obtain information about a list of exercises and then choose a particular exercise on that list. That exercise is linked to one or more examples (or no examples, if there are no examples defined for that exercise). An exercise is also linked to one strategy that uses a ruleset. The ruleset contains rules for operations that can be applied to exercises. The rules are used by the inner loop for analyzing student steps. Exercises are also directly linked to the ruleset. A ruleset is linked to one or more rules. The third line in each dark blue box shows the name of the data type that is used for resource representations in the prototype (e.g., IExample).

Figure 8 shows that there are resources that do not use all the REST methods. The strategy resource, for example, does not use the methods PUT, POST and DELETE, because we do not want the situation that a strategy resource is deleted or that a strategy is created by the REST layer. This would break the restriction that an exercise always has exactly one strategy. When building a REST interface for distributed expert knowledge modules, it is therefore important to analyze the restrictions of such REST interface; what should it be able to do and what should it not be able to do.

8. Prototype
In this chapter we describe the prototype of a REST layer for a distributed ITS. The prototype has been developed using the Servant framework [10] and we used an expert knowledge module built with the IDEAS framework [32] to work with this REST layer. The reason for building a prototype is to demonstrate that it is possible to use the REST architectural style for distributed ITSs. This experimental research consists of two parts:

i. Building an expert knowledge module using the IDEAS framework (expert knowledge modules are called domain reasoners in the IDEAS framework).

ii. Building the REST layer in Servant for the expert knowledge module we built.

In the next sections we describe why we selected the Servant framework and how we developed the REST layer.
8.1 Frameworks for building a prototype

In this section we give an answer to research sub question 5: Are there frameworks that can be used for building a prototype using the IDEAS framework? We found several Haskell frameworks for building REST interfaces. For building the prototype we considered three Haskell REST frameworks: Servant [10], Yesod [35] and Hapstack [36]. A full review of all existing Haskell REST frameworks and a full analysis of which framework to use is outside the scope of our research. We have chosen Servant to build the prototype, because the software engineer only has to focus on the business logic and Servant takes care of the routing [10]. Another advantage is the general-purpose generic programming technique of Servant.

IDEAS is a generic framework for expert knowledge modules. It is not developed for just a specific educational domain, but for a broad range of educational domains (such as Math, Logic, et cetera) [7]. The main idea of Servant is that the web APIs can be interpreted in different ways and that this API language can be augmented with new constructs [10]. The general-purpose generic programming technique of Servant therefore fits well with the generic vision of the IDEAS framework. If the IDEAS framework is extended with new techniques, the REST interface built with the Servant framework can be extended as well to work with the new IDEAS techniques.

8.2 Expert knowledge module for simple calculations

In our prototype we want to provide resources to learning environments in a RESTful way. This section gives an overview of an IDEAS expert knowledge module and describes which parts the REST layer uses for providing the resources to learning environments.

For building and testing the prototype we need an IDEAS expert knowledge module. Instead of using an existing IDEAS expert knowledge module, we built our own expert knowledge module for simple calculations using the IDEAS framework. We could have used an existing IDEAS expert knowledge module, but building our own has some benefits:

- It enables us to learn how the IDEAS framework works in practice.
- Since this expert knowledge module is small, we can quickly adapt it for testing purposes.

This expert knowledge module consists of a domain reasoner that contains some exercises and provides a number of (standard) feedback services [32].

```haskell
dr :: DomainReasoner
dr = describe "domain reasoner for calculations" (newDomainReasoner "eval"
  { exercises = [Some minimalExercise, Some basicExercise,
                 Some evalExercise, Some bonusExercise]
   , services = myServices
  })
```

DomainReasoner is a type defined in the DomainReasoner module of the IDEAS framework. The DomainReasoner dr in the code snippet has four different exercises: minimalExercise, basicExercise, evalExercise and bonusExercise. The type of the exercises is the Exercise type from the IDEAS framework.
evalExercise :: Exercise Expr
evalExercise = emptyExercise
  { exerciseId = describe "Evaluate an expression (full)" $
    newId "eval.full"
  , status = Experimental
  , strategy = evalStrategy
  , prettyPrinter = show
  , navigation = termNavigator
  , parser = readM
  , equivalence = withoutContext eqExpr
  , ready = predicate isCon
  , examples = level Easy [expr1] ++ level Medium [expr2] ++
               level Difficult [expr4] ++ level VeryDifficult [expr8] ++
               level VeryDifficult [expr9]
}

We take the code snippet of the evalExercise exercise to highlight some features of the IDEAS Exercise type. It consists of a record with several fields that we can use to get the resources for the learning environments. In the prototype we use the field ‘strategy’, ‘status’ and ‘examples’ to get the resources we need. Also the function ‘ruleset’ defined in the IDEAS Exercise module is used. This function gives the rules the exercise uses in providing the inner loop functionality.

In our expert knowledge module we have defined rules for adding, negating, multiplying and exponentiation. The strategy is the set of rules that can be applied to some exercise. We also constructed examples of exercises. In the next code snippet we show how such an example is constructed in our expert knowledge module.

def expr4 :: Expr
expr4 = Negate (Add (Add (Negate(Con 1)) (Con 8)) (Negate(Con 1)))

In this code snippet we see that rules for negating (‘Negate’) and adding (‘Add’) have been applied. The Con rule tells that the expression after Con is a constant integer. The expression should be read as: -((-1 + 8) + (-1)) = -(7 + -1) = -(7 - 1) = -6

8.3 REST layer
The Servant framework provides Haskell web API types for describing URIs, representations and methods. It also provides a Server type that drives the routing and can be used as a webserver. The next code snippet is taken from the prototype and gives an example of a web API implementation.

def type ReasonerAPI = Capture "exerciseid" String :> ExerciseAPI
1  :<|> "exercises" :> Get '[HTMLLucid] ExerciseList
2  :<|> Capture "exerciseid" String :> DeleteAPI
3  :<|> Capture "exerciseid" String :
4  |> ReqBody '[JSON] String :> AddExampleAPI
5
def type ExerciseAPI =
6  "examples" :> Get '[HTMLLucid] IExamples
7  :<|> "strategy" :> Get '[HTMLLucid] IStrategy
8  :<|> "status" :> Get '[HTMLLucid] IStatus
9  :<|> "ruleset" :> Get '[HTMLLucid] IRuleSet
10
def type DeleteAPI = Delete '[HTMLLucid] ()
11 def type AddExampleAPI = Post '[JSON] ()
The ReasonerAPI type defines endpoints that are used as URIs. When running the REST layer, the endpoint in line 2 can be queried by using the URI: [http://localhost:8081/exercises](http://localhost:8081/exercises). This URI will return the ExerciseList resource. This ExerciseList resource is a Haskell data type that is used by a function that queries the ‘exercises’ field of the domain reasoner. We also see that only the ‘GET’ method is allowed for this endpoint and that the representation is in HTML. We also could have used JSON as representation or both.

The ‘Capture’ in line 1 indicates that this URI needs the exercise id from the learning environment to get the appropriate resource for returning one of the three endpoints in the ExerciseAPI type (line 5). For example, to get the examples of the exercise ‘evalExercise’, the learning environment must use the URI [http://localhost:8081/eval.full/examples](http://localhost:8081/eval.full/examples). If the learning environment wants the ruleset of this exercise, it should use the URI: [http://localhost:8081/eval.full/ruleset](http://localhost:8081/eval.full/ruleset). Line 3 points to the DeleteAPI (11). This type gives a URI for using the DELETE method on exercises. Since the IDEAS expert knowledge module does not have a feedback service for deleting exercises, we use the REST DELETE method in our prototype to only remove exercises from the ExerciseList resource. Line 4 points to the AddExampleAPI in line 12. This type gives the URI for adding an example to an exercise. A learning environment can add an example to the exercise ‘evalExercise’ by using the URI [http://localhost:8081/eval.full](http://localhost:8081/eval.full) providing the example details using the ‘POST’ method.

Note that the ‘:>’ constructor represents the URI divider ‘/’ and the ‘:<|>’ constructor is used to combine multiple endpoints in a web API type. To give an example: the type ‘ExerciseAPI’ can return three URIs. Those URIs are separated by the ‘:<|>’ constructor. In the code snippet we also see that ReasonerAPI contains all the handlers with a ‘Capture’ in it and that these handlers point to the endpoints in the ExerciseAPI, DeleteAPI or AddExampleAPI. We can see this construction as a representation of the arrows in Figure 8. The ReasonerAPI represents the methods for the ExerciseList resource and from there we can get an Exercise resource by using the ExerciseAPI. The ExerciseAPI contains the URIs and methods for the examples, strategy and ruleset resources. To give an example: the ‘Capture’ in the URI for the examples resource for a particular exercise can be considered as the arrow between Exercise and examples in Figure 8.

To drive the routing of resources to the correct URI we need to use the server type family of Servant. The advantage with the Servant server type is that the software engineer building a REST layer only has to focus on the business logic. Servant server takes care of the routing [10].

```haskell
1 reasonerServer :: IORef DomainReasoner -> Server ReasonerAPI
reasonerServer ref =
  \(\backslash s \rightarrow\) exerciseServer ref (newId s))
:<|> do dr <- liftIO (readIORef ref)
       return (exerciseListResource dr)
:<|> deleteServer ref
:<|> addExampleServer ref

2 exerciseServer :: IORef DomainReasoner -> Id -> Server ExerciseAPI
exerciseServer ref exId =
  do Some ex <- findById ref exId
     return (IExamples (prettyPrinter ex) (examples ex))
:<|> do Some ex <- findById ref exId
       return (IStrategy (strategy ex))
:<|> do Some ex <- findById ref exId
       return (IStatus   (getId ex) (status ex))
:<|> do Some ex <- findById ref exId
       return (IRuleSet  (getId ex) (ruleset ex))
```
3 `findById :: MonadIO m => IORef DomainReasoner -> Id -> m (Some Exercise)`
   `findById ref exId = do`
   `dr <- liftIO (readIORef ref)`
   `findExercise dr exId`

4 `data ExerciseList = ExerciseList`
   `{ drtext :: [String] }`
   `exerciseListResource :: DomainReasoner -> ExerciseList`
   `exerciseListResource x = ExerciseList (map stringFunction (exercises x)) x`

5 `stringFunction :: Some Exercise -> String`
   `stringFunction (Some ex) = showId ex`

6 `deleteServer :: IORef DomainReasoner -> Server DeleteExerciseAPI`
   `deleteServer ref s = liftIO $ modifyIORef ref $ \dr ->`
   `dr { exercises = filter keep (exercises dr) }`
   `where`
   `keep :: Some Exercise -> Bool`
   `keep (Some ex) = getId ex /= newId s`

7 `addExampleServer :: (IsId a) => IORef DomainReasoner -> a -> String -> Server DeleteAPI`
   `addExampleServer ref exS txt = liftIO (modifyIORef ref f)`
   `where`
   `f :: DomainReasoner -> DomainReasoner`
   `f dr = dr { exercises = map g (exercises dr) }`

   `g :: Some Exercise -> Some Exercise`
   `g (Some ex)`
   `| getId ex == newId exS =`
   `| case parser ex txt of`
   `| Left msg -> Some ex`
   `| Right a -> Some ex {examples = examples ex ++ [(Medium, a)]}`
   `| otherwise = Some ex`

8 `run 8081 $ serve reasonerAPI (reasonerServer ref)`

In the code snippet we see in code-section 8 an expression for starting ‘reasonerServer’ with the argument ref. This argument is a reference to the domain reasoner that is using this particular REST layer. We use a reference to a domain reasoner instead of manipulating the domain reasoner itself. The reason for this choice is that it is not possible to store new exercises and examples to an existing domain reasoner. In the deployment phase a way to persistently store exercises needs to be developed (using an exercise database for instance). Developing this data persistency is outside the scope of this research. Therefore we use Haskell’s IORef for testing POST and DELETE. With IORef data is only temporarily stored in the computer’s random access memory.

The reasonerServer is a Servant server that routes the resources to the correct URI in the previous code snippet. reasonerServer in code-section 1 uses other servers to process the resource requests. The exerciseServer (2) is responsible for providing the different resources concerning a particular exercise. It provides the resources ‘examples’, ‘strategy’, ‘status’ and ‘ruleset’ by using the ‘findById’ function (3). This findById function uses the ‘findExercise’ function from the IDEAS framework. These resources are
needed for querying the URIs provided by the ExerciseAPI type. The function ‘exerciseListResource’ (4) is responsible for providing the list of exercises for the ExerciseList resource. This function is used in the reasonerServer (1) and takes a domainreasoner (‘dr’) as an argument. The function deleteServer (6) is needed for the URI that deletes the references from the ExerciseList resource and addExampleServer (7) is used for the URI that adds an example to an Exercise.

Instead of the REST layer invoking the expert knowledge module, we let the expert knowledge module start the REST layer. See code-section 8. The reasonerServer gets as argument the reference to the domain reasoner in the IDEAS expert knowledge module. It is the expert knowledge module that has to import the REST layer. This ‘inversion of control’ has the benefit that we do not have to adapt the REST layer to each expert knowledge module.

9. Conclusion, discussion and future work
To increase the reuse and interoperability of expert knowledge modules, we separated the expert knowledge module from the other three ITS components (the learning environment) and provide it in a distributed way. To provide a standard interface between the distributed expert knowledge module and the learning environment we presented an assessment method for comparing and evaluating architectural styles. This method consists of a table with quality attributes from the ISO 25010 standard that are important for distributed ITSs. Architectural styles can then be assessed by checking to which level they meet each quality attribute. This enables us to select an architectural style, based on its total score, for improving the quality of distributed ITSs. The assessment method also enables us to analyze the trade-offs that come with selecting a particular architectural style. We selected REST from a selection of architectural styles. This choice for REST is based on REST’s highest score and the advantage REST has regarding the evolution of services [20]. We built a prototype of an interface using the REST architectural style.

9.1 Conclusion
The assessment method scores the quality attributes of each architectural style important for distributed ITSs. This provides us with a total score for each architectural style and highlights the trade-offs that come with the choice of a certain architectural style. In other words, with this assessment method we are able to answer the research question: “How can we compare and evaluate REST, Web services and Plug-in architectural styles for the distribution of the expert knowledge module in an intelligent tutoring system?”. Our answer to the research question was found by finding answers to our sub research questions:

RQ1. We made an analysis of what a distributed expert knowledge module is and how it is constructed (chapter 4).

RQ2. By studying the ITS literature we found the quality attributes important for non-distributed ITSs and analyzed how they are affected by distributing the expert knowledge module. With this analysis we made a selection of quality attributes that are important for distributed ITSs (chapter 4).
RQ3. With this set of quality attributes we constructed the assessment method for assessing the Service Oriented Architectural style, the REST architectural style and the plug-in architectural style (chapter 5).

RQ4. We describe how expert knowledge modules can be wrapped up as a distributed component with respect to each architectural style (chapter 6).

RQ5. We selected the Haskell Servant-framework for building a prototype using the IDEAS framework (chapter 8).

We demonstrated that the assessment method helps with finding an architectural style to use for distributed expert knowledge modules by using the highest scoring architectural style for building a prototype. The REST architectural style has the highest score and after analyzing the trade-offs for using REST, we concluded that REST is the appropriate style for building our prototype of a distributed expert knowledge module. However, one of the trade-offs for REST is that interoperability is only ‘partly covered’. We decided that this is not a great disadvantage for our prototype because the negative influence of this quality aspect strongly depends on how stateless the expert knowledge module is. The expert knowledge module we distributed is built on the IDEAS framework and IDEAS is fully stateless [32]. With the prototype we are able to provide an interface between learning environments and a distributed expert knowledge module. We discovered that this interface based on REST provides the standardization we need to increase the interoperability of distributed expert knowledge modules and covers the quality attributes important for distributed ITSs.

9.2 Discussion
We identify three threats to validity in our research:

1. We used existing literature about ITSs and architectural styles to identify the important quality attributes for non-distributed ITSs. The danger with using this literature research is that we cannot be certain that we have all the important aspects of ITSs. We may be overlooking quality aspects that can be important for ITS stakeholders.

2. We did not conduct a full research on who the stakeholders are and what their requirements are. Instead we used the stakeholders mentioned in ITS literature and categorized them under the three user types defined by the ISO 25010 standard. We defined the concerns that are important for the stakeholders instead of conducting requirement elicitation.

3. Although we consider the REST architectural style as the style with the highest score in our assessment method, it is possible that there are architectural styles, not considered in this research, that score higher on suitability for distributed ITSs. Because we have not done a systematic literature review on all possible architectural styles, we may overlook a better scoring architectural style.

The risk caused by the first two threats is that the assessment method may overlook important trade-offs for selecting an architectural style with our assessment method. Threat 1 and 2 have in common that because we may have not defined all stakeholders (threat 2) we are overlooking quality aspects of ITSs that are important to those stakeholders (threat 1). Threat 1 is also caused by the fact that we were not able to compare non-distributed ITSs with existing distributed ITSs.
The third threat is not a threat to our assessment method, but it is a threat of validity concerning our choice for REST in this research. Our choice for REST might not be the best choice if we have overlooked an architectural style that scores better than REST. However, the reason to conduct this research is not to find the best architectural style. Our goal was to develop an assessment method to compare architectural styles against the quality attributes important for distributed ITSs.

Threat 1 and threat 2 are a risk to our assessment method. However, we have mitigated this risk by using existing research on ITSs and research on the distribution of learning environments. This existing research was conducted by experts in the field of ITSs and software architecture. With our assessment method we may not be able to make a detailed analysis on each trade-off when selecting a certain architectural style, but we are now able to make a coarse-grained informed decision on which architectural style to use for building distributed ITSs.

9.3 Future work
An important supplementary question is how to transform our assessment method from a coarse-grained assessment tool into a more fine-grained assessment tool with which we can make detailed analyses on trade-offs. We can split this question into two sub-questions:

i. How can we find all the stakeholders for a particular ITS?

ii. Instead of using stakeholder concerns, how can we find the requirements important for those stakeholders?

The first question requires conducting a full stakeholder analysis for that particular ITS and analyzing what type of stakeholders distributed ITSs typically have. Answering the second question might be a bit more difficult. Since requirements should be measurable, requirements elicitation is needed. One way to do this is by interacting with stakeholders that use a particular non-distributed ITS, the existing system, that will be transformed into a distributed ITS. We leave answering these questions as future work.

We consider the implementation of a REST layer between the distributed expert knowledge module and learning environments also as future work. The prototype described in chapter 8 can be used for this implementation. Because the URIs and resource representations are defined by Servant’s Haskell API types the prototype can be easily extended with URIs to other resources. It is also relatively easy to add new representations to the APIs because Servant allows software engineers to add new content types as resource representations [10]. In the prototype we used the REST methods POST, DELETE and GET. Since Servant follows the HTTP protocol [10] the prototype can be extended with the other HTTP methods.

A point of attention is how to use methods that change the domain reasoner (methods such as POST and DELETE). In the prototype we used IORef to show the effect of these methods. To implement the prototype into a live environment it needs to be investigated how data can be persistently changed in a domain reasoner. A possible approach is to extend domain reasoners with new feedback services that can store data such as examples into a database system. This stored data can then be manipulated by methods such as the DELETE method. We have not investigated if this approach is feasible or if there are other methods. We leave this investigation of changing data in a domain reasoner as future work.
Bibliography


Other resources


[34] web site: http://www.openmath.org/, OpenMath standard, visited on 14 April 2016
