

MODELING COMPLEX SERVICES IN AN E-COMMERCE BROKERING SYSTEM

Luis Bellido, Enrique Vázquez
Technical University of Madrid, Spain
ETSI Telecomunicación, Ciudad Universitaria, 28040 Madrid
lbellido@dit.upm.es, evazquez@dit.upm.es

Francisco Valera
University Carlos III, Madrid, Spain
Avda. de la Universidad, 30; 28911 Leganés, Madrid
fvalera@it.uc3m.es

ABSTRACT

This paper describes a brokering platform for the provision of complex services over the Internet. The platform is able to analyse a request for a complex service, divide it into simple components, and combine several services and products, offered by different providers, into a solution that satisfies the complex request. The platform helps the user to refine or change the request until a suitable solution is found. After the user agreement, it confirms the transaction to the selected providers. Overall, the client can obtain a complex service, involving several suppliers, in the same way as a simple one. The main innovations of the platform are the use of ontologies to handle the static and dynamic aspects of complex services in different business areas, and the control of the transactional properties of complex services. The system exploits advanced technologies, such as multi-device publishing based on XML, mobile agents, the Enterprise JavaBeans architecture, and the Java 2 Platform, Enterprise Edition (J2EE).

KEYWORDS

Service brokering, Service Composition, Ontology, E-commerce, Distributed Environment, J2EE

1. INTRODUCTION

The provision of complex services that require the co-ordination of different providers is a difficult task, especially if it has to be performed automatically by a brokering system negotiating with several e-commerce servers over the Internet. This is the objective of the project Smart-EC¹ (Support for Mediation And bRokering for Electronic Commerce), part of the European R&D Programme on Information Society Technologies (IST).

The project has successfully implemented two broker prototypes, and has demonstrated their capabilities in a scenario of complex publishing services, such as printing a book, creating a customized publication, a multimedia course, etc. Each of them may include several simpler services, for example creating content, translation, cover and page design, printing, and binding. If the customer submits a complex service request that specifies the desired attribute values (page size, paper quality, number of copies, total price, etc.), the Smart-EC system is able to divide it into several requests, one for each simple service that will be needed, with attribute values derived from those specified by the customer. Then the system searches the best offers for each service, and, if the customer accepts, it contacts each selected provider to get all the components needed to fulfill the customer's request. For each attribute, the customer can specify a priority level and a range of acceptable values. The system grades available service offers depending on how well they match the values given by the customer, or, if no suitable offers are found, indicates to the customer which attribute values should be changed. If possible, the system proposes

¹ <http://www.telecom.ntua.gr/smartec/>

advantageous alternatives to the customer, for example a service with better characteristics than the ones requested at the same price, or a good offer that a provider has announced to start in a few days.

The intelligent brokering functions in Smart-EC are based on two key elements. Firstly, a general model of services that includes concepts like customer, provider, attribute, value, request, offer, order, etc., and the logical relations among them. Secondly, an ontology [Uschold, 1996] that describes a particular domain. For example, the ontology for publishing services mentioned above describes concepts like book, author, language, translator, page, binding type, printing technology, etc., and their relationships. The system design is very general, and new ontology modules can be added in order to use it in other domains. Using the service model and the specific knowledge captured in the ontology, Smart-EC is able to assist customers throughout the steps required to fulfill their needs: definition of a request, division into simpler services, search for suitable providers, evaluation of offers, and the final transaction in which all needed components are booked.

The paper is organized as follows. Section 2 gives an overview of the Smart-EC system architecture. Section 3 presents the general service model used, and section 4 describes the scenario of publishing services selected for the demonstration. The conclusions are summarized in section 5.

2. ARCHITECTURE

Figure 1 shows the general architecture of the Smart-EC system. The Smart-EC system architecture is supported by the Java 2 Platform, Enterprise Edition (J2EE) and by the Enterprise Java Beans (EJB), a main component of J2EE [Kassem, 2000]. J2EE and EJB provide a series of system-level services to support applications that can be developed, deployed, and executed in a distributed environment, facilitating the re-utilization of software components.

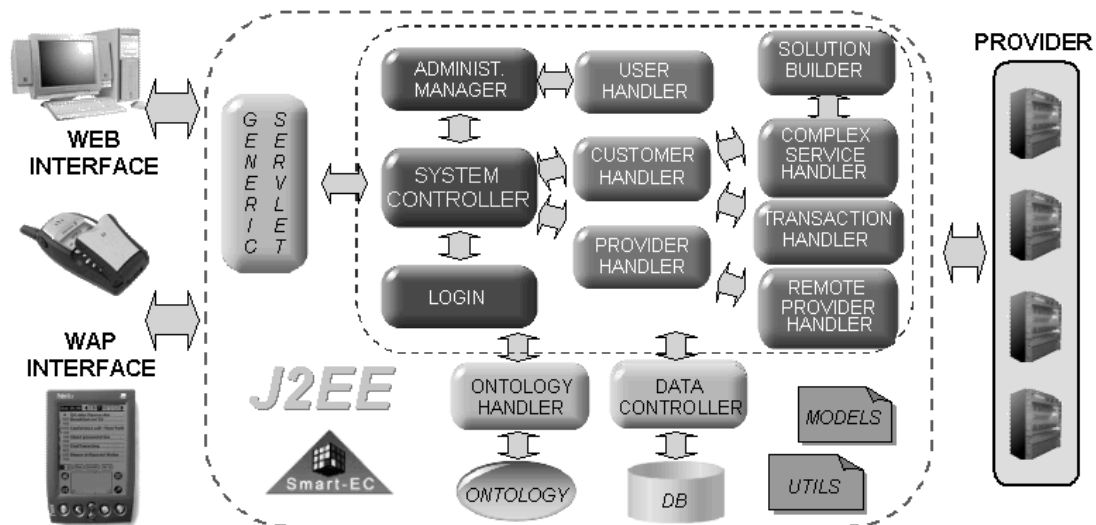


Figure 1: Smart-EC system architecture

The *Generic Servlet* module is in charge of the user interface to access the system, and uses messages in XML (eXtensible Markup Language) [Bergholz, 2000] to communicate with the *System Controller* module. The use of XML to represent the information is combined with stylesheets written in XSL (extensible Stylesheet Language) to define the format in which the information is presented to the user. This XML/XSL combination allows the system interface to be generic and easily configurable for different types of terminals, both fixed and mobile. Adapting the interface to a new type of terminal is reduced to designing a new XSL stylesheet.

While the *Generic Servlet* module task is just the presentation of information, the *System Controller* controls the logic aspects of the access to the system, that is, what options are offered to a user depending on the previous actions of this user.

The *Customer Handler* module guides a customer in the process of making a request. The customer has two options: make a new request or select one request from his/her repository of requests (modifying some values). In order to make a new request, the customer will be able to navigate through the catalogue of services, which is displayed as a tree. If a complex service is selected the customer can go down the tree to see its components. Alternatively, the customer can define a request for the complex service by filling the values of the complex service attributes, and the system will transparently break it down in other simpler services. The *Customer Handler* relies on the *User Handler* to display the tree of services, and the *Ontology Handler*, which is in charge of the access to the ontology.

The *Provider Handler* is in charge of the communication with the providers and it relies too on the *User Handler* to display the tree of services. A provider is able to navigate through the tree of services and register *offers* for those services that are interesting for him/her, in a similar way as a customer can navigate and select services to make requests. Normally, the registration of offers will be done by automatic procedures between the providers systems and Smart-EC. The project has distinguished several types of providers with which Smart-EC is able to communicate [Vázquez, 2001]. The *Remote Provider Handler* manages the connection to remote providers to check the availability of offers on line and to update their catalogue of registered offers.

The *Complex Service Handler* module manages the aggregation and decomposition relationships in the process of building a solution. It takes the request that the customer has built with the help of the *Customer Handler* and uses the *Ontology Handler* to break down or/and aggregate the services in the request. The results are one or several requests that the *Complex Service Handler* will then pass to the *Solution Builder* module, which is responsible of looking for the offers that match each request

The *Solution Builder* looks for the registered offers that better suit a request. This is basically done by comparing the values of the attributes defined in a request to the values that are defined in the offers registered for a given service. Some of the attributes are generic (for example, price or duration), while others are specific to each service (for example, source and target language, or subject of the text, for a publishing translation service).

The process of comparing requests with offers is quite flexible (see 3.1). The system will return a solution consisting of offers which will have a grade depending on how well they verify the conditions of the request, imposing more restrictions to the attributes that are more important for the user. The system can also suggest the modification of the requests, according to the registered offers, so he/she can find advantageous offers that would not be presented for the original request.

It is also possible for the users to select, at the time they make the request, if the system should contact the provider to verify the availability of an offer, that is, if the provider did not run out of stock. While this may increase the delay for obtaining results, it avoids the presentation of offers that would need to be discarded lately. For this task, the *Solution Builder* relies on the *Remote Provider Handler* module, which is in charge of the communication with the remote providers.

Last, the process of selection and grading of the offers takes also into account the user preferences, which can be explicitly specified in the users' profile (for example, preferred providers), or learned by the system from previous interactions with the user.

Once the *Solution Builder* selects the more adequate offers, they are combined to make up one or more *solutions* that are proposed to the user. The presentation of the solutions will depend on the user preferences, for example, only the best solution or several solutions ordered from better (larger grade) to worse, the level of detail of the list of solutions, etc.

When the found solution or set of solutions is not satisfactory, the user can modify the conditions in the request and repeat the process of offer searching. When an acceptable solution is obtained, the user can choose it and request the system to execute a transaction in which all the providers involved in the provision of the complex service are contacted. The *Transaction Handler* module is in charge of reserving the service form each provider. If any of the reservations fail, the rest of them are cancelled.

The architecture of the system is completed by the following modules: *Ontology Handler*, in charge of the access to the ontology, *Data Controller*, in charge of accessing the database of the system (Oracle), *Login*, in charge of verifying the user identity, *Administration Manager*, managing the user accounts and service definition, and *utils*, implementing general functions. Finally, the *Models* module defines the data structures, which are discussed in the next section.

3. COMPLEX SERVICE MODEL

Throughout the system description presented in section 2, different concepts have been informally introduced in order to model complex services in Smart-EC. Basically, there is a catalog of services and every service is defined by attributes with a certain set of possible values. A service may be recursively split in simpler services. Users define service requests by means of conditions that must be compliant with their attribute values. Providers may define service offers in the same terms, setting several conditions over attribute values.

The request for a complex service is broken down into requests for simple services, and different offers are searched for each one. Selected offers are scored and combined in order to form one or several solutions for the demanded complex service. If one particular solution is accepted, the system executes the corresponding transaction between itself and all the providers of the different offers that are included in the solution, preserving the atomicity of the complex service.

Figure 2 represents a set of services with a tree structure. It does not mean that the service catalogue managed by the system is internally organized as a static tree. The figure only represents the view of a group of services that is being dynamically built and is being progressively presented to the user whenever different services are selected (in the prototype, the tree is represented through different menus).

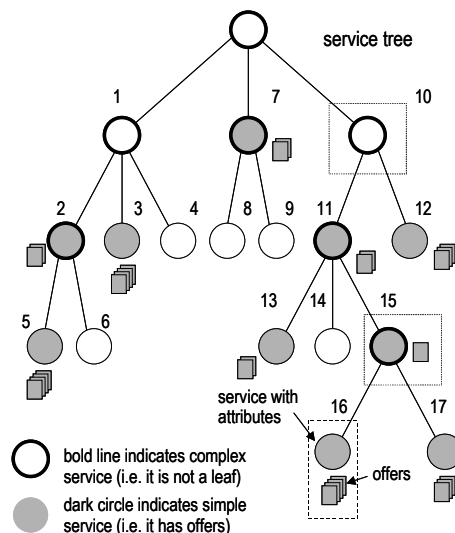


Figure 2. Complex and simple services

Every node of the tree, except the ones that are leaves, represent a complex service, because it is possible to break them down into simpler services (their children nodes). On the other hand, every node with a related registered offer, represent simple services, that is, there are providers that are able to offer the service as a whole.

This way of defining types of services in the model of Smart-EC implies that a service may be simple and complex at the same time: simple because someone is offering it as a whole and complex because it can be broken down, if it is wanted, into other simpler services offered by different providers. For instance, services 2, 7, 11 and 15 in the previous figure may be treated both as simple or complex services. Services 1 and 10 however, can only be treated as complex and the rest of the nodes are only simple services (they cannot be broken down).

The root node of the tree is not really representing a service, but a family of services (electronic publishing, travels, etc.).

3.1 Requests

A request defines a service (complex or simple) that is wanted by a user specifying for every attribute:

- zero or more values

- a condition over the values
- a certain weight or priority
- the type of the attribute

The request is also including:

- a list of preferred providers
- the number of desired solutions
- the search mode (fast or exhaustive)
- the possibility of checking for offer availability before solutions are returned to the users

Every component in a request is optional. If no attribute value is given, the system will understand that any value is acceptable by the user. Possible conditions over the specified value or values include “*equal*”, “*different*”, “*greater than*”, “*less than*”, “*in range*”, “*out of range*”, “*any*”, “*none*”. Through the combination of one or more values and the proper condition, the user may express quite flexibly the properties of the demanded service. For a numeric attribute named “x”, the user may specify for example:

x = 10 (value 10, condition “equal”),
 x ≠ 25 (value 25, condition “different”),
 x = 1 or 2 (values 1 and 2, condition “any”),
 x < 50 (value 50, condition “less than”),
 1 ≤ x ≤ 5 (values 1 y 5, condition “in range”), etc.

It is also possible to define a set of values for attributes of enumerated types. The same as it is done with requests specification is done with offers. An offer for a service (see section 3.2) specifies values for its attributes. If the set of values included in an offer and a request have a not-null intersection, this offer is said to be compliant with the request for the considered attribute.

In his requests, the user says how many solutions must be returned by the system (obtained by the combination of offers for the different demanded services) and how its conforming offers must be searched:

- exhaustively, if the user wants every existing offer to be reviewed (retrieved, scored and compared), or
- as fast as possible (fast mode), if the user just wants the first found compliant offers

The priority of an attribute specifies the relative importance of offer compliance for the user regarding to that particular attribute (“*compulsory*”, “*important*”, “*normal*”). These priorities are used in order to match request attributes with offer attributes. A *compulsory* attribute in a request will force the corresponding attribute in the offers to be fully compliant in order to consider this offer into the set of possible results. In the case of *important* attributes (and even more in the *normal* ones) it is not required that the requested attributes have an exact match with the attribute of the offer (the user may be requesting a hotel by less than 30 euros, but the system may also be considering hotels for 35 euros, for example).

If a request specifies a list of one or more preferred providers, for example a user that prefers to fly with a particular company, this list is used to give a higher preference to the offers by those providers.

3.2 Offers

An offer defines the service as it is given by a specific provider. As it has been said, offers and requests are defined setting values and conditions for service attributes. This is why both structures are very similar in Smart-EC service model. For every attribute of the service, an offer includes

- zero or more values
- a condition on these values

together with:

- the validity period of the offer
- the activeness of the offered service

The possibility of using values and conditions for attributes in the offers, allows that a provider can register with just one offer several services that are only different in the value of an attribute, for example a service to which the provider wants to offer several options with the same price, validity period, etc.

For an offer to be applicable for a certain request, apart from complying with the values of the demanded attributes, it must be verified that the offer is *active* when the request is performed and that the request is made within the time period defined in the offer. This is needed since a provider may decide to temporarily invalidate an offer or to register an offer for upcoming dates, so that users may see offers that are not valid yet. It must also be taken into account that if the request has the “*check availability*” option

set, providers will be contacted in order to confirm offer availability before any consideration about attribute matching is made.

The duration of a service is one attribute more and it is checked with the same constraints as the others. For example, an offer for a binding service that specifies “duration = 3 days” would be valid for a request demanding “duration < 5 days”. The system allows the definition of different units for the values of attributes and the conversion functions between units. This way, the previous offer would also be considered valid for another request with “duration < 1 week”. As any other attribute, duration is optional and it can be useless for some services. In those cases, checking if the request is within the validity period of the offer should be enough. If a service has duration but the user is not including a condition on it in the request, it just means that he will accept offers with any duration.

The price of a service is considered the same way: an offer will be valid (from its price viewpoint) if it is specifying a value within the requested range (taking also into account priorities, as it has been explained).

The Smart-EC model is designed for services, but products can also be expressed with this model without a great effort. A product is described by a set of attributes and both requests and offers may be registered for it. In the case of a product, attributes like “duration” may not be needed (although some others like “time to deliver” should be defined).

The previous example of the 3 and 5 days, would be equally applicable to the “time to deliver” or the price (changing the corresponding unit), or in fact, for any other attribute.

The Smart-EC service model and the prototype developed within the project allow the management of any attribute, without having to consider its particular meaning. If a request is demanding a set of values (numeric or of any type) for an attribute “x”, the system will just search offers where this “x” attribute is included with a set of values whose intersection with the request set of values differs from null (providing unit conversion if it is necessary). The user may express its preferences concerning the importance of each attribute through the mentioned fields of weight and priority.

In summary, this approach facilitates the definition of products and services with new attributes and their inclusion in the system database without having to rewrite all the methods that compare requests and offers. Reasonably, the role of defining and consistently integrating new services within Smart-EC catalogue will belong to the system administrator. Users and providers will be restricted to the possibility of requesting or offering existing services.

3.3 Solutions

A solution is just a combination of offers, generally coming from different providers that the system is returning as an answer to the request for a complex service.

As it has been already mentioned, a user can ask the system for several solutions for the same request. If one of them is accepted, the system starts the execution of a transaction that provides the complex service atomically, so it will provide every included simple service or none of them. Transaction management details are not treated here, but they can be consulted in [Vázquez, 2002].

One of the most interesting things about proposed solutions is the added value that the system is providing in order to help users to find better offers. For every solution, the system will mark (and typically show with a different color) attributes that are not fully compliant or not compliant at all (their priorities in the request were set to “normal” or “important”), so that users can easily detect non-matching attributes. The solution usually includes additional information in order to know:

- if the offer has been checked online for availability
- if the whole offer is compliant or not (taking into account all its attributes)
- if the offer is orderable without any other restriction
- attributes that should be changed in order to get better offers

4. DEMO SCENARIO

The first prototype of the Smart-EC system has been demonstrated in a scenario of simple and complex publishing services, defined by a publishing company that participates in the project. The particular

hierarchy of services is represented in Figure 3. The attributes of each service, possible values, units, etc. are defined in detail in [Ruiz, 2001].

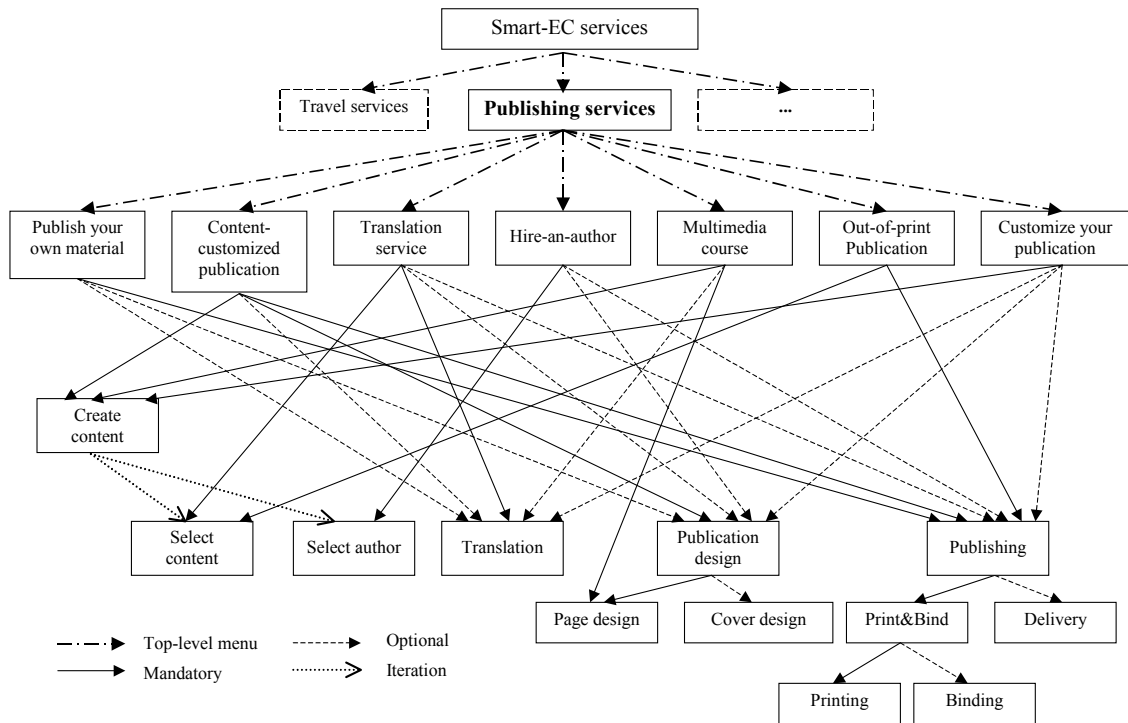


Figure 3: Scenario of services in the publishing sector

4.1 The ontology

The ontology included in the system prototype describes in a more formal way the elements that are found in the selected scenario and the logic relationships that exist between them, for example, what simple services make up a complex one.

Ontologies can be used in electronic commerce systems to facilitate the mutual understanding between buyers and sellers, or to facilitate information re-utilization. Some intelligent search engines use Ontologies to find documents containing terms that are logically related to the term input by the user (even terms and relationships initially unknown by a user who is not an expert about the subject he/she is looking information for) [Debnath, 2000].

As it was previously mentioned, the main purpose of the ontology in Smart-EC is to provide a complex service globally, building it from a combination of simpler services that are obtained from different providers. The Smart-EC ontology, written in a language derived from Classic [Borgida, 1989], allows the definition of services, such as the complex service *Publish your own material* (see Figure 3). This service allows a user to edit his/her own books from his/her own material, including services for translating, design, printing and binding. The Smart-EC ontology also defines the attributes of each service, and the constraints and possible values for each attribute. For example, attribute *price* is constrained to be greater than zero and, for *Publish your own material*, attribute *cover material* can have one and only one of the following possible values: *paperback*, *hardcover*, *leather*, *imitation leather* or *cloth*. Finally, the Smart-EC ontology defines the relationships between the values of the attributes in a complex service and the values of the same attribute for the simple services that compose it, such as how the price for a complex service is split between the simple services.

5. CONCLUSION

This paper has described the objectives and design of an e-commerce broker for complex services. The brokering systems available in the Internet often deal with simple services separately, e.g. a hotel reservation, or with a service package offered by one provider. The Smart-EC system is able to split a complex service into simple components, select offers from different providers, and combine them in one transaction. Therefore, with Smart-EC the customer can get a complex service, involving several providers, in the same way as a simple one. In addition, the system helps the customer by evaluating the available offers according to the requirements given in the customer request, and by proposing alternative offers that may improve what the customer requested, for example a better service at the same price, a cheaper service in a slightly different date, etc. These brokering functions are habitually provided by human agents with a good knowledge of the service sector of interest, but are difficult to automate. To achieve this goal, the Smart-EC system uses the generic service model described in this paper, complemented with domain-specific ontologies.

The project has implemented and demonstrated two prototypes that include an ontology for a realistic scenario of publishing services, also described in the paper. These two prototypes have been evaluated by a user group in order to validate the conformance with user requirements and to assess the concepts of mediation and brokering. The results of the analysis by the users are positive: a platform that offers one single complex global solution, constructed from basic services, adapted to the user's specific needs would allow a position in the market based on differentiation. A major concern for the users, such as the possibility of customisation and adaptability to the user's needs is solved by the generic service model defined in Smart-EC, though future work will be needed to facilitate the definition of specific ontologies for new domains.

In general, the objectives and approach of Smart-EC are in line with the recent proposal of evolution towards a *Semantic Web*², in which the information has a well-defined meaning and can be processed automatically in order to improve the interactions between humans and computers in the Internet.

ACKNOWLEDGEMENT

SEMA Group sae (Spain), Technical University of Madrid (Spain), National Technical University of Athens (Greece), Laboratoire d'Informatique, Robotique, et Microélectronique de Montpellier (France), France Télécom R&D (France), AQL (France), Tradezone International Ltd. (UK), and Anaya Multimedia (Spain) participated in the Smart-EC consortium.

REFERENCES

- Bergholz, A., 2000. An XML Tutorial. *In IEEE Internet Computing*, Vol. 4, No. 4. pp. 74-79.
- Borgida, A., 1989. CLASSIC: A Structural Data Model for Objects. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Portland, Oregon, USA. pp. 59-67.
- Debnath, S. et al, 2000. LawBot: A Multiagent Assistant for Legal Research. *In IEEE Internet Computing*, Vol. 4, No. 6. pp. 32-37.
- Kassem, N., 2000. *Designing Enterprise Applications with the Java™ 2 Platform, Enterprise Edition*. Addison-Wesley.
- Ruiz-Andino, A. et al, 2001. Publishing Services Demonstration. <http://www.telecom.ntua.gr/smartec/>
- Uschold, M. and Gruninger, M., 1996. Ontologies: Principles, Methods and Applications. *In Knowledge Engineering Review*, Vol.11, No.2. pp. 93-136.
- Vázquez, E. (editor), 2001. *Smart-EC Architecture version 1*. Deliverable 4.1. <http://www.telecom.ntua.gr/smartec/>
- Vázquez, E. (editor), 2002. *Smart-EC Architecture version 2*. Deliverable 4.2. <http://www.telecom.ntua.gr/smartec/>

² <http://www.w3.org/2001/sw/>