Architectural abstractions for space awareness

Francesco Tisato¹, Daniela Micucci¹, Marzia Adorni¹, Eric Cirasa¹

¹D.I.S.Co. – Università degli Studi di Milano – Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milan, Italy
{tisato,micucci,adorni}@disco.unimib.it

Abstract. It is widely known that the recognition of a sound software architecture makes the success of a software system. Unfortunately, even if the domain model relies on sound ontological bases, its mapping into an architecture that effectively reifies the model is a difficult task. In order to fill the semantic gap, relevant domain concepts should directly turn into architectural abstractions. Space plays a crucial role in many application domains but, surprisingly, no related architectural abstractions have been defined yet. The paper proposes space-related architectural abstractions derived from the application of classical software engineering principles. Since similar principles supported the definition of space ontologies, the conjecture is that our space architectural abstractions are good candidates for a formalisation in ontological terms.

1 Introduction

Crucial aspects in the development of a complex IT system are the identification of both a sound domain model - possibly derived from an ontology-based approach - and a sound architectural model.

Software architecture “refers to the structure of the system quite hidden from the user” [17]. Then it captures relevant issues that are loosely related to domain issues. This results into a wide semantic gap between domain models and architectural models. Even if the model of an application domain relies on sound ontological bases, its mapping into an architecture that effectively reifies the model is a difficult task.

In order to fill the semantic gap, relevant domain concepts should directly turn into architectural abstractions i.e., architectural patterns and components.

In particular, this holds for the concept of space, which plays a crucial role in many application domains. Surprisingly, in the software engineering area do not exist any space-related architectural abstractions even if, since millennia, philosophers and scientists deal with spatial ontologies. On the contrary, space-related issues are usually intermixed with domain-specific and implementation-dependent issues.

The identification of “orthogonal” architectural abstractions that explicitly model the space is still an open issue, whose solution should rely on the conceptual framework provided by spatial ontologies [1] and by research achievements in several application domains like GIS [16], surveillance [9] and robotics [3].
Generally, a software architect identifies architectural abstractions by relying on “architectural commonsense” i.e., by exploiting principles of the software engineering like separation of concerns, encapsulation and multiple views. Under this perspective, it is reasonable to apply the architectural commonsense to identify spatial architectural abstractions.

Conversely, concepts close to the architectural commonsense have driven the definition of space ontologies. Consequently, our conjecture is that spatial architectural abstractions may provide useful guidelines for the definition of related ontologies.

The objective of this paper is to contribute to the cross-fertilization between ontologies and software architectures by focusing on the concept of space. Section 2 reviews established software engineering concepts and their relationships with relevant issues in the spatial ontologies area. Section 3 sketches some architectural abstractions that could help building space-aware systems. The conjecture is that such abstractions are good candidates for a formalisation in ontological terms. Finally, Section 4 sketches some conclusions and tries to stimulate the discussion among different communities.

2 Software engineering vs. spatial ontologies

This section reviews some established sw engineering principles and their relationships with emerging issues in the spatial ontologies area. Nothing new here – the aim is to highlight the similarity of concepts used in the ontologies and in the sw architectures areas. A comprehensive review of spatial ontologies can be found in [1].

2.1 Separation of concerns

Separation of concerns [14] refers to the ability to identify, encapsulate, and manipulate only those parts of software that are relevant to a particular concept, goal, or purpose. Concerns are the primary motivation for organizing and decomposing software into manageable and comprehensible parts [21]. Applying separation of concerns to the concept of space means, for instance, that the intrinsic features of a car and of a person should not depend on their locations. On the other side, it is useful to know if there is an entity at a given location – no matter whether it is a car or a person. Moreover, the distance between two locations does not depend on the presence of entities.

Separation of concerns has been recognised as a key issue by the spatial ontologies community too. As pointed out in [15], there is lack of both theoretical and technical frameworks that explicitly describe the fundamental spatio-temporal characteristics in a context-independent way. [20] introduces the three domain model, which carefully separates theme, space, and time. [13] highlights the need for a clean separation between entities and locations. [1] outlines that equating objects with their locations is at least problematic, and that the particular location of an object is not a necessary ontological feature of that object. [16] exploits relationships in the thematic domain to indirectly associate spatial properties with entities.
2.2 Encapsulation

Encapsulation is a basic requirement to ensure separation of concerns. Encapsulation means that the focus is not on “what” an object is, but on “how” it behaves from the point of view of an external agent – be it a human or a program. Object-orientation reifies this concept by separating the interface of a class from its implementation. In layered systems a higher layer exploits functionalities exported by a lower layer without knowing its internal structure. In service-oriented architectures the requestor of a service is not aware of how it is implemented. Several ontological approaches seem to endorse the concept of encapsulation. The focus is not on “what” a space is, but on “how” we can reason about a space – for example, by defining the set of performable queries about locations. This approach is consistent with the database-oriented attitude of the GIS community; [16] defines a comprehensive set of operators on spatial databases. Other approaches focus on how an object can move around the space. In [5] spatial relations are derived from schemes of action. In [2] “the structure of space... is measured by how we can move in it and how objects can be placed within it”, according to the Gibsonian concept of affordance [8].

2.3 Multiple perspectives

The capability of viewing and organising entities under different perspectives is another aspect of separation of concerns. This is a widespread concept in relational and multi-dimensional databases and, in general, in software engineering. Complex systems require several space models to co-exist. Possibly different models correspond to different levels of abstraction. For example, a high-level spatial model of a building can be a graph where each node is associated with a room and an arc is associated to a passage; this level of abstraction may support the strategic planning of a robot motion as a path over the graph. On the other hand, the fine-grained trajectory of the robot is defined in a Euclidean space. The same entity (a robot) must be located in both spaces: once the high-level path of the robot has been planned in terms of a sequence of nodes, it must be mapped into Euclidean coordinates in order to define the fine-grained trajectory. Therefore there is the need for a mapping between the two spaces.

Often the concept of space must be extended to model conceptual spaces and their mappings. For example, the localisation of an employee office may require a mapping between an organisational space (the firm organisation chart) and a graph space (the firm offices).

Significant research activities on ontologies deal with multiple perspectives and, in particular, with multiple spaces. The realist perspectivalist approach [18] embraces, not one ontology, but a multiplicity of complementary ontologies – distinct perspectives on reality, each one of which is veridical. [9] discusses different kinds of spaces in video-surveillance systems, mappings between spaces, and knowledge representation that is to be used by another computer program. [10] shows how multiple interacting, qualitative and quantitative, topological and metrical, spatial representations can be integrated in a common framework that can express states of
4 Francesco Tisato1, Daniela Micucci1, Marzia Adorni1, Eric Cirasa1

partial knowledge. [1] highlights that there may not be a single ontological perspective that is sufficient for all tasks, so that a foundational ontology needs to provide for the different perspectives that can be taken on phenomena in the world; within Geographic Information Systems, a range of perspectives has always been required and a major issue is that of reconciling these perspectives and providing mappings across them; there are substantially different ontological domains which need to be maintained separately; re-use and modularisation are inevitable for an effective treatment. There are several proposals of stratified and tiered ontologies, in which various ontological perspectives may be organised and related; see, for instance, [4] and [7]. [6] exploits the concept of levels of abstraction and suggests the use of semantic translators as a powerful solution for interoperability.

3 Spatial abstractions

This section discusses how basic SW engineering principles – separation of concerns, encapsulation and multiple perspectives – can be exploited to design spatial abstractions supporting, on one side, the development of well-structured space-aware application components; on the other side, the seamless integration of application components, which rely on different spatial models.

The architectural abstractions are described in UML. Though a discussion on UML is out of the scope of this paper, it should be pointed out that the formal definition of UML2 [18] and the Model Driven Architecture approach [15] provide a level of soundness that allows UML2 to be considered as a formalism for the definition of spatial ontologies.

3.1 Subjective spaces

Fig. 1 sketches how classes that model separated concerns can be grouped in packages. A space-aware application reasons in terms of thematic entities, defined in a thematic domain, which can be located in a spatial domain. The abstract Entity class is the superclass of subclasses that model space-independent thematic concepts and are visible to the applications. Pedestrian, Car, Vehicle, are only some examples and are grouped in the thematicDomain package. Components dealing with spatial issues have no visibility over such subclasses; they are just aware of the existence of generic entities modelled by the Entity class.

A spatial domain defines the concept of generic subjective space (Fig. 2), which is a container of locations. A location is an opaque handle allowing a generic entity (i.e., an instance of Entity) to be located in a space. A generic subjective space provides methods for defining locations, for anchoring entities to locations, for getting the entities at a given location and for getting the location of an entity. Note that in principle such general methods do not depend on a specific model of the space; therefore GenericSubjectiveSpace is modelled as a concrete class.
Architectural abstractions for space awareness

Fig. 1. Separation of concerns

Fig. 2. Spatial domain

By itself, a location is just an opaque handle allowing thematic entities to be anchored somewhere. Of course, a location must embed the concept of “where”, which cannot be defined without relying on a specific space model. The idea is that the semantics of any space model is defined in an operational style through the concept of movement. A movement (Fig. 3) models an elementary operation allowing a location to be reached from another location. In general, a location is associated with (at least) one movement stating how the location can be reached from a reference location.

A movement is defined by specifying a couple of locations and some movement parameters. The abstract class MoveParameters is specialised according to specific space models: for example, in a Cartesian 2D space the movement parameters are a couple \{dx, dy\} of real numbers. In a simple graph space the movement parameters are empty, as a movement just models an arc between nodes i.e., locations.

There is no need for the explicit definition of all the possible movements between locations – which would be impossible in a continuous space. A movement can be
defined whenever it is useful; in general, a location must be reachable via at least one associated movement in order to capture the concept of “where”.

Note that the concept of “where” is recursively defined by stating how a location can be reached from another one via a movement. To terminate the recursion, a subjective space may define an origin i.e., a fluctuating location providing the starting point for the definition of other locations.

A subjective space (Fig. 3) is a container of locations which embodies the semantics of a specific space model. The definition of a movement is made via the defMovement() method that encapsulates model-dependent issues. In particular, the implementation of the method must ensure that the definition of the movements is consistent with the space model. For example, in a Cartesian space it makes sense to state that each location has a basic associated movement defining how it can be reached from the origin, and that any other movement must be consistent with the basic one. On the other side, in a simple graph space any movements (i.e., arcs) can be freely defined.

In general, a distance can be defined for a movement. The concept of distance is modelled by the abstract class Distance, whose subclasses define the measurement unit for a specific space model. Applications may ask for a movement distance via the distance() method defined in the SubjectiveSpace class.

Finally, a path is an ordered sequence of one or more movements between two locations. The identification of all the available paths between two locations is made via the path() method in the SubjectiveSpace class. The length of a path is the sum of the distances of its movements. The distance between two generic locations can be defined as the length of the shortest path between them.

3.2 Integration: space service

Subjective spaces are defined from a space model. When applications define their subjective spaces from the same space model, the semantics of every location is
Architectural abstractions for space awareness

understood by each application. Interoperability is guaranteed if a *shared space* is defined as the union of the locations defined in several subjective spaces.

Fig. 4. Space services

Fig. 4 sketches how separation of concerns is maintained with the introduction of the shared space concept that is captured by the `genericSpaceService` and `spaceService` packages. The former one embodies a shared generic space, the latter a shared space whose semantics relies on a specific space model.

The consistency among the (generic) subjective spaces that form a shared space can be ensured if each (generic) subjective space delegates the execution of its operations on locations to a *generic space service* (Fig. 5). Consequently, the `GenericSpaceService` class provides all the methods defined in the `GenericSubjectiveSpace` class.

Fig. 5. Generic space service

All the applications that need to share a space must exploit the same `GenericSpaceService` (the same instance in O-O parlance) via their own `GenericSubjectiveSpaces` (Fig. 6).
Applications sharing a space

The SpaceService class (Fig. 7) is a subclass of GenericSpaceService capturing the concept of “where” (embedded into a location). It provides the same methods as the SubjectiveSpace class. Since paths and movements rely on a specific spatial model, SpaceService is an abstract class and its subclasses embed the specific spatial models. Likewise GenericSubjectiveSpace class, SubjectiveSpace delegates the execution of its operations on path and movement to SpaceService class. Since the semantics of the space model is now managed by the service, SubjectiveSpace becomes a class concrete.
3.3 Space mapping

Locations defined by different space services are based on different space models. This implies that an application, whose subjective space relies on a space service, is not able to manage locations defined by another space service. This raises a difficult architectural problem i.e., how to integrate different spatial models by preserving the separation of concerns.

Conceptually, locations mapping is the process of obtaining a new location from a location defined by a different space service. MappingService class (Fig. 8) embodies the mapping process. Since mapping depends on specific space models, MappingService is an abstract class. Its maps() method, given a location defined by the source GenericSpaceService, defines a new location expressed in terms of the space model reified by the target GenericSpaceService. Maps() method invariant is that the source location has been defined by the source GenericSpaceService.

![Fig. 8. Mapping spaces](image)

Fig. 9 sketches an example of mapping usage. Imagine that App1 needs to send loc1 defined over a Cartesian space model to App2 that exploits a geographical space model. App1 relies on the cartesianToGeographical MappingService that defines the function converting a location from a Cartesian space to a geographical space. After receiving loc2, App2 adds the received location to its subjective space.

If there is an entity E anchored to loc1, App1 may anchor E to loc2 before delivering loc2 to App2. The result is that App2 “views” E in its geographical space.
4 Conclusions

The paper presented a set of space-related architectural abstractions arisen from the application of well-consolidated principles like separation of concerns, encapsulation and multiple perspectives. One of the achieved advantages is the re-use. Basic classes encapsulate spatial semantics, which is defined in an operational way. Consequently, the simply specialisation of these basic classes allows specific space models to be defined.

The paper presented the abstractions by highlighting the similarity of concepts exploited by both the ontologies and the sw architectures areas. Our conjecture is that the proposed abstractions may provide useful guidelines for the definition of related ontologies. Our intent is to stimulate the discussion among different communities.

Finally, it is worth noting that separation of concerns, encapsulation and multiple spaces are properly exploited by software platforms aimed at supporting the development of advanced graphical applications. For example, in Java3D [19] there is a clean separation between spaces and domain entities. There is a hierarchy of spaces. Moreover, the location of an object relative to a space is defined in terms of a transformation of coordinates which models a movement. This encourages us to develop a framework based on our abstractions.
References


