Earth Sci Inform (2009) 2:299–313
DOI 10.1007/s12145-009-0034-7

RESEARCH ARTICLE

A skeleton design theory for spatial data infrastructure

Methodical construction of SDI nodes and SDI networks

Rolf A. de By · Rob Lemmens · Javier Morales

Received: 7 July 2009 / Accepted: 25 September 2009 / Published online: 12 December 2009

Abstract In this paper, we look into the theory of designing geoservice systems, i.e., SDI networks and their constituent SDI nodes. As the field of SDI is strongly about bridging between geoservice systems, interoperability and harmonisation, it is not surprising that standardisation efforts are of crucial importance in it. These efforts have historically addressed abstract and concrete content models for data and metadata exchange, as well as abstract and concrete behavioural models for computational processes. The list of standards that are in use in the SDI field continues to expand, and reaches out to neighbouring fields such as sensor nets. We argue that given these trends, the resulting levels of standardisation in actual systems, and the complexity of geoservice systems in general, it appears only natural to look into the possibility to define a *standardised design theory* for SDI and its nodes, which addresses the function base and the communication base. Specifically, we provide an overview of those components that need to be designed, and what are their relationships. We do so in an abstract way, focussing on the concern of information content in this paper, and only hinting at an appropriate theory of realisation based on our skeleton theory.

Communicated by: H.A. Babaie

R. A. de By (✉) · R. Lemmens · J. Morales
ITC, Enschede, The Netherlands
e-mail: deby@itc.nl

R. Lemmens
e-mail: lemmens@itc.nl

J. Morales
e-mail: jmorales@itc.nl

Keywords Catalog · Design · Formal method · Geoservices · Implementation · Spatial data infrastructure

Motivation and paper organisation

Our ever changing world is one with growing concerns for sustaining the human and natural condition, as well as one with growing potential for spatial monitoring and understanding of ongoing earth processes in the natural and man-made environment (Davis et al. 2008). One of the important tools of our time to understand these spatial processes and exploit spatial understanding is spatial data infrastructure (SDI).

It needs to be emphasised at the same time that SDI must be seen as a, still maturing, enabling technology with a somewhat unspecified client base. One important and potentially large application domain is that of societal networks: communities of (groups of) civilians, public administrators and professionals that share information on a theme of interest. Such theme my have societal relevance in health, education, crime, public administration practice, or environmental surveillance. Land use change in the Amazon Basin is a point in case, as it draws (and requires!) attention from myriad stakeholders. Green pressure groups may identify disallowed sugar cane plantations or may want to report illegal logging. The SDI may provide the base data to back up their claims, for instance by putting on record the land cover history of the disputed locality. We expect that SDI used in this sense supports transparency of the management of public space.

We consider SDI enabling because it can provide the geospatial substrate on which relevant social functions

can be based (Davis et al. 2008). For instance, community members may want to share news, experiences, recommendations and even advice, and may want to associate any of these with precise or even uncertain spatial tags. It follows that models for network member communication are important to develop, and should be addressed in appropriate and well-understood protocols. In this sense, SDI provides one of the important bases from which to develop community support systems.

In context of the philosophy of single-creation–multiple-use (Bernard et al. 2005), the SDI domain is well-known for its relentless focus on the application of standards (Crompvoets and Bregt 2003), and of ontologically steered mapping mechanisms (Arpinar et al. 2006). The two enable the exchange of data, computations, and computation outcomes—all of those seen as services—between constituent systems of the SDI, which in the remainder of this paper are called SDI nodes. The terms *SDI* and *SDI node* are defined below.

The standardisation efforts have historically addressed abstract and concrete content models for (spatial) data and metadata exchange, as well as abstract and concrete behavioural models for (spatio-) computational processes. The list of standards (in use) continues to expand, and reaches out to neighbouring fields such as sensor nets.

Given these efforts, the resulting levels of standardisation in actual systems, and the complexity of geoservice systems in general, it appears only natural to look into the possibility to define a standard design theory for SDI and its nodes. After all, given the high incidence of adoption of, often the same, standards, hopes can be had that even the realisation of the systems can be made subject to some level of standardisation. This paper makes an attempt to break into this area. We believe that it is important to do so as an accepted design theory will emphasise the need for high-level, platform-independent designs that over time will lead to more robust and better understood system implementations.

To us, an *SDI node* is a moderately to highly complex information system with usually a long lifetime expectancy, in which geospatial resources feature rather prominently. Its design and subsequent implementation approach should respect this expected lifespan, in terms of abstraction, documentation and formal transformation and derivation. We argue that SDI node design and implementation itself should be a domain of methodical standards and mappings, both of which must be understood deeply, and applied rigorously.

We define an *SDI* as a *collaborative network of system and human actors that exploit contributed data and computational resources, many of which are spatially explicit, for one or more targeted objectives, making use of service offerings and consumptions*. An SDI functions as an organising mechanism for data, computational models and classification schemes across data themes and application domains, tackling issues of identification, interoperation, and exploitation of spatial resources held by its community. SDI nodes are the system nodes of the SDI network, which is completed by non-system nodes, of which humans are the most important to mention here.

In a not too distant future, SDI is expected to shape as a support technology for systems of a more argumentative kind, in which stakeholders can express their views, backed up by 'spatial proof' or 'spatial annotation'. It is also in this light that we argue for the importance of communication as a fundamental SDI characteristic.

Our definitions are intentionally ignoring the non-technical characteristics of SDI, not because they are unimportant, but because they receive appropriate attention elsewhere (de Man 2006; Masser 2005; Williamson et al. 2003), because they are not subject to design scrutiny, and also because they are difficult to quantify in an explicit, formal design framework that is the subject of this paper.

Present-day practice executes SDI node design and realisation in a rather ad hoc and sometimes even unstructured way. There are multiple reasons for this. First of all, SDI development is a new field nestled in a relatively new engineering domain (Craglia et al. 2008; Vieira et al. 2008). Secondly, a design theory for GIS-centric systems is lacking, certainly when compared with the design theory for database-centric systems. In the latter, much more standardisation has taken place between different platforms, not in the least because of the earlier standardisation work on SQL and derivatives (Codd 1970; Astrahan and Chamberlin 1975), leading to the present situation in which abstract, platform-independent designs are rather easily realisable.

Approaches to realise SDI can be either top-down, bottom-up, or by-accident. *Top-down* approaches often address a wide thematic coverage for a larger administrative entity, being goal-unspecific, addressing all organisational (legal, financial, institutional) matters and can be potentially client- or market-ignoring. These systems are true SDIs-by-design, but their number is small (Bernard et al. 2005; Annoni et al. 2008). Bottom-up approaches often derive from a well-understood clientele and their service demand, for a relatively small

thematic and/or geographic coverage. These systems have been realised in some numbers (Masser 1999; Lance 2003; Williamson et al. 2003). Finally, SDIs-by-accident arise typically where legacy systems have been found to hold exploitable spatial resources and have been made to deliver such as services. A number of these systems has been realised as well. The latter two approaches constitute, in our opinion, a rather natural evolutionary development suiting the field of SDI best.

Another reason for the somewhat immature state of SDI engineering lies in its mix of system paradigms. Historically and effectively, SDI is much about *sharing and exchanging data*, notwithstanding the fact that other service promises may include offerings to use compute power, storage space, run an application, help to find services, or orchestrate a new service from existing ones. So, SDI can be firmly placed in the domain of data and information systems, and thus fits in the realm of a *state-based paradigm*. This is actually more true of SDI node design than of design of SDI as a network, which brings us to another paradigm. If an SDI constitutes a network, that network needs to be designed, which materialises to the precise specification of forms of communication—in protocols, interfaces, services—between actors in the network. That is, we need a *communication process paradigm*. Thirdly, the SDI field associates strongly with principles of the geosemantic web, which claims potential to develop services to its expanding clientele that were not foreseen at design time, and will be generated on the basis of deep (spatial and thematic) understanding of needs and context. We call this the *introspective paradigm*.

Yet another reason for the lack of a fundamental SDI design theory hides in the rich and growing spectrum of implementation platforms (database engines, image repositories, GIS server engines, metadata engines, middleware and portal technology) that can be used for SDI implementation. Relative luxury though this may seem, and notwithstanding the massive effort in standardisation of recent years, these systems are not based on a single theory of spatial information. This causes the need to implementors to be aware of all sorts of idiosyncracies between these systems and their models, and in our view, forces them to focus on those details far too early in their design and implementation work.

Overseeing the above, a necessary step, in our view, is to add formal rigour to the design and implementation process for SDI systems. We argue that such a rigourous, formal approach to SDI node development is naturally state- and function-based as nodes captures spatial resources such as data sets, spatio-computational models, and so on. A formal approach to development of SDI as a network , in contrast, is naturally process-based. This leads to the situation that an SDI node must display two formal faces: state and functionality on the one hand, and process behaviour on the other hand. We also argue that, given current technology, the introspective paradigm to SDI design is best handled as a third phase in the design. In this way, almost naturally the system extendibility and the required system abstraction is addressed last.

Our approach to SDI design is presented in Section "What constitutes an SDI design philosophy?", providing a somewhat axiomatic statement plus arguments of what we believe is needed for the field. A theoretic framework for state- and function-based SDI design is subsequently presented in Section "The vertical perspective: developing state-and-function systems". A similar framework for communication-based design is described in Section "The horizontal perspective: developing a communicating system of systems". The interplay between the two is discussed in Section "Merging horizontal and vertical designs". Achieving system introspection and orchestration by design is the topic that we address in Section "The introspective perspective: developing an adaptive and extensible system of systems". How such designs can be properly integrated with both the vertical and horizontal perspective is discussed in Section "Merging in the introspective design".

What constitutes an SDI design philosophy?

It is a justified question whether the field of SDI is in need of a separate design theory. After all, distributed information systems (DISs) have been in place for a few decades already, and one can argue that an SDI is just that. One may not be able to pinpoint a single design theory for large-scale DISs, but the scientific community certainly has produced a few results in the method and formalism domain for their design.

Yet at the same time, SDI is sufficiently distinct from DIS, warranting special treatment in its design as well. We recognise at least the following important distinctions:

Geospatial data characteristics Geospatial data forms the fuel of any SDI, and comes with (quality) characteristics that make its direct application often non-trivial. Typical DIS mostly operate on a principle that the system state represents the world-as-we-know-it, while such an assumption can be disastrous in many use cases in SDI context.

Infrastructural foundation An SDI intends to provide a data sharing framework for multiple purposes, such as the economy, the environment and society at large. So SDI is strongly aimed at enabling 'other business'. Typical DISs target a single business, or at least a confined set of use cases.

Use case understanding Where a typical DIS is often developed for a number of well-defined and well-understood use cases, for instance fitting inside a single enterprise perspective, the use case understanding for an SDI is often partial at the start only, and assumptions are made that other promising uses will develop over time.

Organismal growth In line with above remark, SDI does not typically develop from a master plan with defined outer boundaries, but rather may start small and dedicated and evolve over time into wider directions that were unforeseen at the start. This comes naturally with all of the above distinctions.

Because system development is an expensive process, appropriate documentation is beneficial when system redesign is called for. It also pays off if the system needs to demonstrate some level of introspection, for which see below. An important abstraction mechanism in IT system design is the move away from platform idiosyncracies. Provided that standards adherence is well established between platforms, as is the case with DBMSs, and more and more also with geospatial platforms, a platform-independent design will always pay off in the long term.

It is through *transformational design techniques* that abstract designs can be methodically and correctly mapped onto a platform of choice. Such steps are illustrated as arrows in Fig. 1. In that figure, the top level represents abstract designs that are platform-independent. A design theory should include formalised transformations that turn abstract specifications eventually into robust implementations. Such transformations are not deterministic functions, as they will need to accommodate design alternatives along the way.

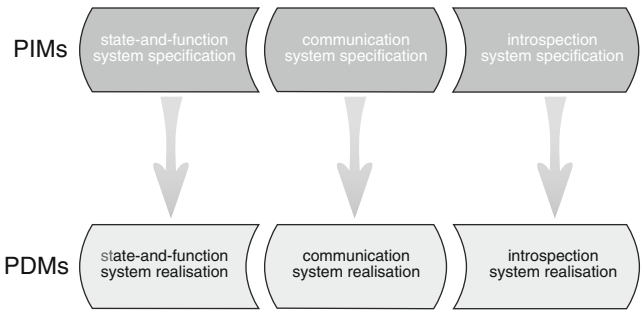

Fig. 1 Overview of SDI design philosophy, showing state/funtion systems, communication systems and introspection system, at both platform-independent and platform-dependent levels

Current SDI initiatives are strongly focussed on data provision, sharing and portrayal, guided by the principle of 'create-once-use-many-times'. We expect them to become much richer functional entities, which, in search of a wider clientele, will need to provide means of *introspection*. The publish-find-bind paradigm is a first step in that direction, as it makes a small step in the domain of metadata exploitation. Introspection can only be achieved if the system does not only provide access to its data, but opens up selected parts of its structure, which is fundamentally captured in a design process.

The methodical design of a complex system, or system of systems, should only be undertaken through a carefully selected list of concerns, which are addressed independently from each other. A classical listing should contain the following concerns: information content, information structure, performance, distribution, interaction, functionality and security, though more could be identified. In the present paper, we focus on only two: information content and interaction. We also leave aside transformation techniques in this paper.

The sequel of the paper discusses abstract design only. We do not here discuss mapping such designs onto platforms, though this obviously required. In the sections, we aim at identifying the components that an abstract design should contain, and the relationships that exist between them. This is done in the following sequence:

1. **State-and-function systems**, in which we focus on how to specify an SDI node as a system that holds information, and provides functions that operate on that information. We call this the *vertical perspective* because such systems are realised often in a software stack.
2. The **Communication system**, in which we focus on how to specify the communication patterns between SDI nodes, abstractly constructing a larger system, the SDI. We call this the *horizontal perspective* to emphasise peer SDI nodes communicating with each other.
3. The **Introspective system**, in which we focus on augmenting the specified SDI system with means to query itself for service possibilities and allow the dynamic creation of new services.

The vertical perspective: developing state-and-function systems

In the vertical perspective, we focus on how to develop a single SDI node. Eventually, its role in the SDI is that of service provider and consumer, but to get there it needs to hold and maintain resources such as data and computational sources, as well as facilities to offer and consume services.

Preliminaries

In designing and realising state-and-function systems, one typically defines a state space and a function collection. In so doing, a prominent tool is a collection of syntactic types $\mathscr{T}$ and a collection of syntactic expressions $\mathscr{E}$. These allow us to make explicit system structure, function signatures, and eventually function definition. We do not elaborate on possible choices for $\mathscr{T}$ and $\mathscr{E}$ here, except to indicate some minimal requirements that would need to be met. A full definition of $\mathscr{E}$ would include variables, constants, identifiers, and an array of operators and functions for manipulation of values. Such a definition would be followed by the definition of the 'well-typed' relation $:\subseteq \mathscr{E} \times \mathscr{T}$ indicating which expressions are properly typed, and what their type is.

On the types collection $\mathscr{T}$, we will assume the presence of a fixed set of *base types*, which might include `void`,[1] `int`, `real`, `bool`, `geometry`, and `image`. For what follows, we also must assume the presence of a syntactic name set $\mathscr{A}$, members of which we will be using as attribute names in records and record types. From types in $\mathscr{T}$, we will be allowed to construct more complex types through the use of *type constructors*. The following type constructors are at least present, and provide an inductive, albeit partial, definition for $\mathscr{T}$:

[function type constructor] $\forall \sigma_1, \sigma_2 \in \mathscr{T} \Rightarrow (\sigma_1 \to \sigma_2) \in \mathscr{T}$

[set type constructor] $\forall \sigma \in \mathscr{T} \Rightarrow \mathbb{P}\sigma \in \mathscr{T}$

[pair type constructor] $\forall \sigma_1, \sigma_2 \in \mathscr{T} \Rightarrow (\sigma_1, \sigma_2) \in \mathscr{T}$

[record type constructor] $\forall i | a_i \in \mathscr{A}, \sigma_i \in \mathscr{T} \Rightarrow \langle a_1 : \sigma_1, \ldots, a_n : \sigma_n \rangle \in T$

[1] The type `void` represents the empty type; it is included here for purely technical definitional purposes only. The type has only one value, $\perp : \texttt{void}$ (where $\perp$ is pronounced as "bottom").

An expression of function type is a function, where σ_1 denotes input type and σ_2 the output type, respectively. An expression of set type $\mathbb{P}\sigma$ denotes a set, with members of type σ. The pair type constructor comes with a special equivalence axiom:

$$(\sigma, \texttt{void}) \equiv (\texttt{void}, \sigma) \equiv \sigma,$$

for any type σ.

Proper states

An SDI node, at a sufficiently abstract level, can be characterised by a state space S and a function space F. We can characterise both as follows. The data store of the node, holding amongst others all the spatial data sources, is seen as a complex object with data type σ. The definition of σ is the field of spatial data modelling, and we will discuss this in some more detail below. Values s of type σ are possible states of the data store, and because it is typically unacceptable to allow all such values, during the data modelling phase, we identify constraints $\gamma_i(s)$ that the stored data should obey always. The conjunction $\bigwedge_i \gamma_i(s)$ of all these constraints is called γ here.

This leads us to a definition of the system *state space S*:

$$S \stackrel{\text{def}}{=} \{ s : \sigma \mid \gamma(s) \}. \tag{1}$$

The state space is the collection of allowable system states. All the designer's work is in providing proper definitions of σ, the *database structure*, and γ, the *static constraints*. From these, the state space S follows.

If the state space of our SDI node is managed as a spatial database, σ classically is a record type of which the attribute names function as table names, and the types associated with those names are themselves set-of-record types. But this is just the classical set-up, and we will discuss extensions appropriate to SDI nodes below in Section "The introspective perspective: developing an adaptiveand extensible system of systems".

Proper dynamics

In a similar vein, the function space for the SDI node must be seen as a collection of functions that encapsulate the complex object, and that provide access to it. Such access can be for information retrieval, updates, or combinations thereof. This, obviously, is a rather classical view of an information system. To formalise this perspective on our system, we need to define the possible dynamics of the system, both in how it replies

to external requests as in how it reflects requests to update itself.

For the time being, we adopt a simplistic model for capturing allowed dynamics of our SDI node, in the sense that we define only a model for allowed one-step changes. In SDI design practice, this model often suffices in capturing most of the dynamic constraints.

This leads to the definition of the system *state change space* U:

$$U \stackrel{\text{def}}{=} \{\ (p, q) : (\sigma, \sigma) \mid (p, q) \in S^2 \wedge \phi(p, q)\ \}. \qquad (2)$$

The state change space is the collection of allowable system state changes, indicating that a state p can turn into state q if and only if $(p, q) \in U$. The formalisation amounts to providing the definition of $\phi(p, q)$. Similar to γ for static constraints, $\phi(p, q)$ is a conjunction of separate dynamic constraints $\phi_i(p, q)$. Each of these expresses a specific condition that restricts the new state q depending on old state p.

Proper functions

The definition of U serves as a backplane of proper behaviour for a collection of functions F that are yet to be defined. These functions must obviously be well-behaved with respect to S and U. These conditions are captured in the definition of function space F, in which we assume that σ is the defined state structure (as per above):

$$\begin{aligned} F \stackrel{\text{def}}{=} \{\ & f : (\sigma, \sigma_{in}) \to (\sigma, \sigma_{out}) \mid \\ & \sigma_{in}, \sigma_{out} \in T \wedge \\ & \forall s_i, s_o : \sigma, t_i : \sigma_{in}, t_o : \sigma_{out} \mid \\ & s_i \in S \wedge f(s_i, t_i) = (s_o, t_o) \\ & \Rightarrow (s_i, s_o) \in U\ \} \end{aligned} \qquad (3)$$

A function f is in the function space F if and only if

- it takes as input an allowed system state $s_i : \sigma$ and additional input $t_i : \sigma_{in}$ from which it produces an allowed output state $s_o : \sigma$ and additional output $t_i : \sigma_{in}$, and
- the one-step state change (s_i, s_o) is allowed.

Any function meeting these conditions is an allowed function. In practice, a system will offer to its outside world a finite collection of functions, known as the system's *interface* $I \subseteq F$. Functions in I fall into two categories: *queries* and *updates*. Queries are special in the sense that they will not (ever) change the system state. A function $f \in F$ is a query if and only if:

$$\begin{aligned} & \forall s_i, s_o : \sigma, t_i : \sigma_{in}, t_o : \sigma_{out} \mid \\ & \quad f(s_i, t_i) = (s_o, t_o) \Rightarrow s_i = s_o. \end{aligned}$$

Finally, we remark that by virtue of the conclusion $(s_i, s_o) \in U$ in the definition of F, and the definition of U, we can conclude that an allowed function f can only return an allowed system state $s_o \in S$.

Spatial data modelling

Spatial data infrastructures are about sharing geospatial data and computational resources, but we have not stated much about the geospatial domain yet. This is obviously important. In this short section, we look at the geospatial dimension of state-and-function system design.

Spatial data modelling is conceptual data modelling for information systems, with specific attention for the assignation of spatial semantics to conceptually identified information content. In plain terms, this means that we attribute some object classes with one or more spatial characteristics.

For example, creeks may be associated with a linestring attribute, or under a more detailed model with a multilinestring attribute, representing all waterways is a creek watershed. One might even consider to additionally assign a polygon attribute to delineate that watershed as an area. Observe that the above affects our definition of σ in Eq. 1.

Attribution of spatial characteristics to object classes is only a starting point. Under the state-and-function paradigm, we continue with identifying constraints to be expressed over our spatial attributes, as well as over combinations of related spatial attributes.

Continuing from the previous example, a multilinestring model for a creek should have all segments connected, while its polygon attribute should allow only hole-free polygons, by nature of watersheds. Moreover, the (multi)linestring representation should be spatially included in the polygon representation. Observe that this all addresses the γ part of Eq. 1.

On the function side, finally, we will need to devise spatial functions on our information system that accommodate information needs of the end-users, and functions that robustly allow spatial data updates within the system. We hope to address the design method for such functions in an upcoming paper.

As to spatial update functions, for example, one function that allows adding a segment to the creek waterway multilinestring will need to verify that the added

segment is actually connected to the old multilinestring. Moreover, in presence of the polygon representation, that polygon may require expanding to continue satisfying the topological constraint of spatial inclusion. Observe that this part addresses which functions f we will have in collection F of Eq. 3.

Outlook to the horizontal perspective

The vertical perspective defined and discussed above considers an SDI node to be a complex object system that holds a state, and offers functions operating on the state. At some point, we assume that all the SDI nodes have been defined in this way.

We continue in the horizontal perspective to look at how the SDI nodes can effectively form an SDI. Crucial to the community of nodes is a proper definition of possible forms of communication between them. To this end, we apply a process algebraic formalism, such that we can describe the SDI nodes and their role in that communication process, effectively describing allowed 'conversations' and appropriate exchange of information.

This brings us to a new paradigm where the things of study are *processes*, not states or functions.

The horizontal perspective: developing a communicating system of systems

Proper behaviours

Under the horizontal perspective, we define what are allowed forms of communication, or 'interactional behaviour', between nodes in the SDI network. The SDI nodes are considered independent systems with their behaviour defined in a declarative way. Such a declarative definition is what we call a *process specification*.

In the formalism that we adopt here, a process specification defines a set of possible behaviours, in which a behaviour is a sequence of actions, ordered in time. An *action* is either an observable communication or it is an (unobservable) internal event of the node, viewed as a process. A communication indicates a *channel* over which the communication can take place plus a communication parameter as placeholder for the communication content.

Each SDI node is thought to exist in a world of its own, and all that it exposes is its behaviour as *services*. Services manifest themselves as *offerings* or *consumptions*, and a typical scenario is that one SDI node offers a service that at some point in time is consumed by another SDI node. In that service consumption event, information may be exchanged between the two nodes. Expanding from the above, we add that human users can be service consumers also, and that more than two nodes/users can be involved in a single service consumption. Moreover, the metaphor of service offering-and-consumption is an intuitive one fitting service-oriented philosophy, but some of these events are better viewed as the synchronisation of actions that take place at different systems, involving information exchange. In other words, there may not be so clearly visible roles of consumer or offerer.

Single services, not necessarily offered by the same SDI node, can be strung together to form a *service chain* that satisfies requirements stated by the designer. In this context, a service chain represents some intended process logic.

Where service, process, action, channel and communication parameter are terms of the specification world, the terms *service execution*, *process execution*, *process step*, *connection* and *communicated value* are respective instantiation terms in the real-time execution world. A service execution within a chain is called a *process unit*. Observe that service chains cannot normally be uniquely assigned to a single node, and are better viewed as 'behaviour of the SDI network'. Intuitively, a service chain is a designed collaboration between SDI nodes.

The behaviour of a service chain has the following principles:

- connections among process units exist only momentarily; as a result of the process unit execution, other connections between subsequent process units may be created,
- accessible resources vary over time,
- process units interact using synchronous communication through interfaces, and
- the basic mechanism of interaction is message exchange.

A family of formalisms that provides significant expressive power to capture these notions is that of *process algebras* (Bergstra and Klop 1987), with the π-calculus (Milner et al. 1992; Parrow 2001) being one specific formalism. We have adopted it in this paper, as it has been shown to be extendable and it allows dynamic adaptation of the process over time, which appears potentially suitable for applications displaying a.o. introspection. Ongoing research links the π-calculus with languages for web service specification, such as WS-BPEL, WS-CDL and XLANG, as a semantical foundation (Lucchi and Mazzara 2005; Feng et al. 2008).

The formalisation of communication between SDI nodes has been neglected by the SDI development community, resulting in underdeveloped use of SDI systems. The premise here is that a precise definition of the communication logic enables the exploitation of SDI nodes and produces outputs resulting from instantiating process chains that more fully adhere to user requirements. It follows that the proper specification of an SDI from the horizontal perspective requires a two-fold approach.

This puts us into position to indicate what are the design targets on the process communication front, having indicated S, U and F on the state-and-function front.

- For each SDI node i, the process specification B_i specifies the node's communication possibilities. This gives us a collection of process specifications, $\{ B_i \}_i$, one for each SDI node.
- For each service chain j, the process specification C_j specifies the service chain's behaviour. Collectively, we will accumulate a set of such behaviours $\{ C_j \}_j$.

We will add a few comments on these two collections after the technical discussion on process formalism below.

Process formalism

In this section, we discuss process specifications in more depth as they are somewhat unfamiliar in the SDI domain. Key to the behaviour specification of an SDI are process specifications such as B_i indicated above.

Given the principles listed above, we will tacitly assume that members of the set of expressions $\mathscr{E}$, defined above, can be used as names of communication ports, variables or as data values. These expressions allow us to represent inputs and outputs that can be attached as prefixes to a *service specification*. Prefixes stand for conditions associated with the instantiation of a service. A prefix is either an input, output or silent prefix, and always follows the $\alpha . S$ form. Prefixes can take the specific forms shown in Fig. 2.

These prefixes cover the simple conditions that are attached to a unit of behaviour, explicitly specifying actions that precede the definition. This provides us with the basis to present an example of two communicating SDI nodes. Here, on the left we indicate a service consumer (or client), and on the right a service provider (or server). Their communication over a channel c is defined through parallel composition (expressed with $|$, and defined below):

$$\bar{c}req . C \mid c(gcap) . P. \tag{4}$$

This behaviour specification indicates that the consumer can send a request message *req*, after which it behaves as C, where behaviour C still needs to be specified. The provider process accepts a message in placeholder *gcap*, for get_capabilities, and then continues as behaviour P, defined below.

$$P \stackrel{\text{def}}{=} (\tau . BS + \tau . IS).$$

This specification, in turn, states that the provider performs an internal operation, that cannot be influenced externally, represented by τ, and then behaves as a *Busy Server* service *BS* and processes the request. Alternatively, it internally rejects the request and behaves as an *Idle Server* service *IS*. Two extra notational elements appear in these statements:

Parallel Composition $P \mid Q$ denotes processes P and Q executing in parallel, where possible synchronising on common channels. I.e., P and Q can interact during the execution or may as well act independently from each other.

Choice: $P + Q$ denotes a behaviour in which either P or Q is executed.

To illustrate the use of the language, we present the formalisation of a simple but typical example of an OGC Web Service (OWS) interaction, namely a

Output Prefix:	$\bar{c}v . P$	denotes a service that sends a value v along the channel c, and thereafter behaves as P.
Input Prefix:	$c(x) . P$	denotes a service that expects to receive a value x along the channel c before it starts to behave as P.
Silent Prefix:	$\tau . P$	denotes a service that can behave as P without any interaction with its environment. τ represents a silent action, intuitively expressing that no action is required to initiate the behaviour P.
Replication:	$! P$	denotes a service that can be instantiated and then replicates itself for further offering.

Fig. 2 Action prefixes of the applied process algebra

find-bind protocol, in which a user connects via a geoportal to find a geoservice offered by a server, and then consumes it. That server may offer various services. This is an example of a service chain process C_j, as we will discuss in Section "Revisiting the design targets for communication processes". The chain involves three SDI components: the user node, the geoportal, and the server. A specification of all the components of this chain can be formalised as follows:

$$
\begin{aligned}
&OWS\_SERVICE_i() \\
&\quad \stackrel{\text{def}}{=} [gconn][gcap][gserv][cr_i][sr_i](\\
&\qquad USER(gconn, gcap, gserv) \\
&\qquad |!GEOPORTAL(gconn, gcap, gserv, cr_i, sr_i) \\
&\qquad |!GEOSERVER_i(cr_i, sr_i)) \qquad (5)
\end{aligned}
$$

This service chain $OWS\_SERVICE_i$ is defined as the composition of various constituent elements of the desired service. It starts by defining a set of global channels that are used by the different service components to interact. The channel *gconn* serves for user connection and identification, allowing, a.o., the user to identify herself and the geoportal to assign a set of privileges. The channels *gcap* and *gserv* represent the interface the geoportal makes available to service users. They respectively serve to carry a request for metadata (a get_capabilities request) and a request for a specific geoservice (a get_service request). Both channels are therefore passed as arguments to the *GEOPORTAL* and *USER* services. To be clear a *get_service request* represents a service call from a client to one of the operations offered by OGC-compliant services. The channels cr_i and sr_i represent the interface between the geoportal and a geoservice provider. They are also passed as parameters to those respective services.

$$
\begin{aligned}
&USER(gconn, gcap, gserv) \\
&\quad \stackrel{\text{def}}{=} [rc][ro](\overline{gconn}(userid, \ldots) \\
&\qquad\qquad |\overline{gcap}(req\_c, rc) \\
&\qquad\qquad |rc(res\_c).\overline{gserv}(req\_o, ro)) \qquad (6)
\end{aligned}
$$

The *USER* service creates two restricted channels *rc* and *ro* that are used by the geoportal service to send responses to the user. The *gconn* channel is used to create a connection to the geoportal. Once connected, the *USER* service makes a metadata request using the get capabilities channel *gcap* and waits for the response through the *rc* private channel. The metadata request content is represented by the *req_c* parameter. When a response arrives over the private channel $rc(res\_c)$, it is processed and (if applicable) a geoservice request is sent over the channel *gserv*, providing again a response channel *ro*. The parameter *req_o* in the geoservice request $\overline{gserv}(req\_o, ro)$ can be instantiated with any of the operations exposed by OGC compliant services. For example a '*GetFeature*', or a '*GetMap*' or a '*DescribeFeatureType*' operation.

$$
\begin{aligned}
&GEOPORTAL(gconn, gcap, gserv, cr_1, sr_1, cr_2, sr_2) \\
&\quad \stackrel{\text{def}}{=} [rgc_1]rgs_1][rgc_2][rgs_2](\\
&\qquad gconn(userid, \ldots).P\_CONNECT(\ldots) \\
&\qquad |gcap(req\_c, rc).(P\_CAPABILITIES(req\_c, rgc_1). \\
&\qquad\quad \overline{rc}(res\_c) \\
&\qquad |GEOPORTAL(gcap, gserv, cr_1, sr_1, cr_2, sr_2)) \\
&\qquad |gserv(req\_o_i, ro_i).(GEOSERVER_1(req\_o_1, rgs_1). \\
&\qquad\quad \overline{ro_1}(res\_o_1) \\
&\qquad |GEOSERVER_2(req\_o_2, rgs_2).\overline{ro_2}(res\_o_2) \\
&\qquad |GEOPORTAL(gcap, gserv, cr_1, sr_1, cr_2, sr_2))) \\
&\qquad\qquad (7)
\end{aligned}
$$

The *GEOPORTAL* service exposes three processes that are respectively used to connect to the geoportal, to query metadata and to consume a geoservice. Upon receiving a metadata request $gcap(req\_c, rc)$ the geoportal instantiates the capabilities process $P\_CAPABILITIES(req\_c, rgc_1).\overline{rc}(res\_c)$ and creates a duplicate of itself to become available for other requests. The result of processing a capabilities request is returned to the user using the corresponding response channel $\overline{rc}(res\_c)$. Alternatively, when the geoportal process receives a geoservice request $gserv(req\_o, ro)$, the geoserver process is instantiated. The set of pairs cr_i and sr_i represent the channels used by the geoportal to communicate with various service providers. In the specification above, we have depicted two possible geoserver services, $GEOSERVER_1$ and $GEOSERVER_2$. Again the geoportal replicates itself upon receiving a geoservice request, so that it becomes available for further requests.

We continue by specifying one more level of detail for the *GEOSERVER* process. Clearly, a *GEOSERVER* service can evolve into different services, depending on context and inputs. To differentiate between the cases, we use the *Match* (=) operator that allow us to express guard conditions. In our particular example, we assume that *IS* can evolve into different

services, a web coverage service WCS, a web feature service WFS or a web map service WMS, etc.

$$
\begin{aligned}
&GEOSERVER_I(cr_1, sr_1)\\
&\stackrel{\text{def}}{=} \big(cr(req\_c, rgc_1)\,.\,CSW(\ldots)\\
&\quad \mid sr_1(req\_o_1, rgs_1)\,.\,\big([req\_o_1 = \text{'gc'}]\,.\,WCS(\ldots)\\
&\qquad\qquad + [req\_o_1 = \text{'df'}]\,.\,WFS(\ldots)\\
&\qquad\qquad + [req\_o_1 = \text{'gm'}]\,.\,WMS(\ldots)\\
&\qquad\qquad + [req\_o_1 = \ldots]\\
&\qquad\qquad \big).\overline{rgs_1}(res\_o_1)\\
&\quad \mid GEOSERVER_I(cr_1, sr_1)\big) \qquad (8)
\end{aligned}
$$

The geoserver process can be triggered with two different inputs. If the input $cr(req\_c, rgc_1)$ is received, the metadata service $CSW(\ldots)$ is instantiated. When the $GEOSERVER_I$ process receives a geoservice request $sr_1(req\_o_1, rgs_1)$, it checks for the type of input. The process can be invoked with different values. Equation 8 depicts three values: *gc*, *df* and *gm*. The parameter values represent a *GetCoverage*, *DescribeFeatureType* and *GetMap* operation request respectively. The replacement of the given parameter will trigger the instantiation of one of three potential *Server* services *WCS*, *WFS* or *WMS*. Equation 8 also contains the term $[req\_o_1 = \ldots]$, which depicts that a other parameter values can be used. We have purposely omitted some of the detail of the specification. All details of the service can be presented in a full-fledged specification of the service, but this falls beyond the purpose and scope of this paper.

Revisiting the design targets for communication processes

After this discussion of the formalism for communication process specification, we can make more explicit how members of the sets $\{ B_i \}_i$ (the SDI node processes) and $\{ C_j \}_j$ (the service chains; indices j distinct from indices i) can be specified, and how this results in the overall behaviour of the SDI network.

The design targets are the process specification B_i for each SDI node i, as well as the service chains C_j. For either of these processes one typically encounters the specification template

$$Service \stackrel{\text{def}}{=} !\,c(x)\,.\,S_i,$$

where *Service* is either a B_i or a C_j, c is the input channel at which the service is listening, and S_i denotes the actual service process. Service chain processes C_j may be 'hosted' by a separate chaining server.

To do justice to the formalism, we need to also introduce a final notation, known as *guarded generalised sum*, denoted by $\sum$. Its informal definition states that

$$\sum_i \pi_i P_i = \pi_1 P_1 + \cdots \pi_n P_n,$$

where each π_i is an input, output or silent prefix, and each P_i is another process expression. We assume that n is known from context.

This now allows us to finalise the behaviour B of the overall SDI network as

$$B \stackrel{\text{def}}{=} \sum_i \pi_i B_i + \sum_j \pi_j C_j.$$

The prefixes function as guards, allowing to identify the proper subprocess.

A mirror look at state-and-function systems

At this stage, the question is how an SDI node seen as a state-and-function system relates to the same node seen as a communicating process. We provide an intuitive answer to this question only.

If we take another look at the client-server communication expressed in the process expression (4), where c is the name of the channel over which the client communicates with the server, one may look at it as the name of the service offered. That, in a state-and-function system, translates to a corresponding function f being offered in the system's function set F. So, naïvely channel names (like c) correspond with function names (like f), and our design should indicate that correspondence. This can be achieved, obviously, by a naming convention that is enforced in the design method.

Caring to take a somewhat less naïve look, we could functionally interpret the action $\bar{c}r$ as a request for service by the client, thus something that could be realised as a remote procedure call. In the same vein, the action $c(gcap)$ could be read as the reception of a service request by the server.

This example shows that in our specification, we have separated service requests from service responses, the latter not occurring yet in our example. So, this constitutes an important footnote to the one-to-one correspondence of channel names and function names: in the above specifications, the channel c corresponds to the service request (only), and another channel may be associated with the service response. Again, a naming convention can ease the art of specification by linking requests and responses somehow name-wise.

A second footnote is that a single function might associate with different channels, each channel using the

function somewhat differently, for instance, through specific choice of passed-on parameter values.

Outlook to the introspective perspective

Given the above framework, we can in principle design and realise single SDI nodes, so also a community of them, as well as design and realise the communication infrastructure for the community. This typically seals the case.

But there is more to a spatial data infrastructure because the resources available in it (data and related computational sources) come with one important characteristic: the resources' *metadata*. The metadata is critical because it is fundamental to decisions of aptness-for-use of the resources that they describe, whether true data, models or other geospatial services. As a consequence, a service specification may need to display intelligence in exploiting these resources, and change its process structure accordingly. It is for this reason that we also address the need for introspective functionality.

As we will demonstrate in a lightweight mode, our choice of formalism above is an appropriate one to tackle these issues.

The introspective perspective: developing an adaptive and extensible system of systems

SDI nodes and their introspection

A first issue is how a single SDI node can be designed and realised to display also introspective properties. Our approach is to refine the framework of Section "The vertical perspective: developing state-and-function systems". More specifically, we sketch how state-and-function systems can display local introspection.

The fundament for this is to identify descriptive information about the system state and functionality as being part of the state itself. In blunt terms: metadata is just data to be represented inside the system state. This is expressed in

$$\sigma \equiv \sigma_{\text{data}} \oplus \sigma_{\text{meta}},$$

where σ corresponds with same from Eq. 1, and $\oplus$ denotes type concatenation. The type language may require extension to facilitate the definition of σ_{meta}. For instance, one can imagine the inclusion of additional types like `iso19115` (for data), `iso19119` (for services), `WS-BPEL` or `OWL-S` (for service chains), and derivatives (such as iso19115 profiles) of this. Those types will typically take the form of XML schemas, each of which will be subject to a specific constraints.

Further, we distinguish between intrinsic and extrinsic metadata. Intrinsic metadata can be automatically derived from the data structure or service functionality, such as bounding box and attribute type names. Extrinsic metadata are annotations, such as service access point and organisational contact details.

Continuing with the reformulation of Eq. 1, $\gamma(s)$ deserves a similar treatment. A simple syntactic analysis will lead to reformulation as

$$\gamma(s) \equiv \gamma_{\text{data}}(s) \wedge \gamma_{\text{bridged}}(s) \wedge \gamma_{\text{meta}}(s).$$

This expresses that fact that some constraints are pure *data constraints*, a number are *bridged constraints*, and others are pure *metadata constraints*.

Pure data constraints are well-known. Bridged constraints express consistency between stored data and metadata, essentially forcing the metadata to be correct descriptors of the data, as well as of the functions offered by the SDI node. Bridged constraints typically affect only intrinsic metadata. Metadata constraints are those amongst the metadata, some of which are inherent to the metadata type definition (e.g., iso19115), others reach beyond it. They may for example restrict coordinate system expressions (e.g., by valid EPSG codes), data attribute types (based on formal classification), and service quality parameter types.

Just like we detailed the definition of constraint γ above, we could detail out the definition of the dynamic constraint $\phi(p, q)$ of Eq. 2. Interestingly, where dynamic constraints are often uncommon on ordinary data, they are not so much on metadata. Especially when it comes to lineage characteristics one expects that a formulation of $\phi_{\text{meta}}(p, q)$ expresses some form of historic monotonicity on the lineage properties.

After this extension of the state space definition, we turn to look at the functions offered by the SDI node. We use the same distinction as before, and recognise pure *data functions*, *bridged functions* and *metadata functions*.

The data functions operate only on the σ_{data} part of the state, and are the 'routine functions'. More interesting are the bridged functions, which query/update both data and metadata. They are typically needed in two scenarios:

1. When metadata intelligence is required to determine output of the function, for instance, when the metadata is required to determine which data is used for the result;
2. When an update on the data requires the metadata to be updated as well, typically as a consequence

of existing bridged constraints. One can view this class of functions as ordinary update functions that trigger metadata updates.

The metadata functions are also an interesting class. These function operate on metadata only. Again, two scenarios appear to be the most likely:

1. Metadata queries probe the system for its resource characteristics, and form an important foundation for initial communication between systems. One may think of getCapabilities requests, and others in that family.
2. Metadata updates in their pure form are those which affect extrinsic metadata, such as the change of organisational contact details or the change of URL as service access point.

SDI network introspection

With SDI node-specific introspective functionality in place, as per above, these nodes can be addressed for ordinary data and computational services, but also for introspection. This brings *service chaining* and *service orchestration* into view.

In service chaining, services of different providers are joined to form a more encompassing service. The conditions of joining should be verified, and are usually expressed as conditions over the metadata of the service providers. This verification should be done at design time, and lead, where necessary, to run-time metadata conditions. One also expects that the service chain is monitored by a system itself, typically different from the main service providers in play. Such a server might be called a *metaserver*. Metaserver functionality is typically restricted to a subset of SDI nodes. Within the SDI network a hierarchy of nodes must be defined, properly controlling the workflows and dataflows within it.

Things become even more interesting when services within a service chain are not decided upon at time of design of realisation, but are dynamically determined at run-time. In this context, we then speak of *service orchestration*, and in light of the present paper of the design of the orchestration. The metaserver that monitors the (now more dynamic) service chain needs to exploit the full introspective capability of potential service providers to identify one or more realisations of the chain.

It is interesting to note that the π-calculus formalism with which we described communication mechanisms in Section "The vertical perspective: developing state-and-function systems" intrinsically provides for such dynamics. We finish this section with an illustration to that extent.

A typical introspective scenario is followed in the *publish-find-bind* paradigm. Timewise, this should read the 'publish——find-bind paradigm', as publishing happens at a separate point in time. It involves a communication of a provider to register a service with a catalog server.

In the second and third legs, *find-bind*, a catalog service is requested to identify appropriate service provisions for getting the overall wanted service. At this stage, the service that eventually will be consumed is unknown. In terms of our Section "The vertical perspective: developing state-and-function systems" formalism, this means that the channel associated with that eventual service consumption cannot yet be named.

The scenario unfolds for instance in a communication between a *Catalog Server* and an *XService Server*. The communication between the two takes the shape of the following composed process, leading to synchronisation over channel f:

$$\bar{f}b \,.\, CS \mid f(z).\bar{z}a \,.\, XS.$$

This statement expresses that the *Catalog Server* service, having found a candidate service b, sends this information along channel f (for 'found'); the *XService Server* receives the link through channel f and then uses it as a channel itself(!), as expressed in $\bar{z}a$, upon which it behaves as XS. One should understand this behaviour as a variable z that gets instantiated, and is subsequently interpreted as a channel (=service) name.

As a point in case, Fig. 3 illustrates a scenario in which a service chain implements map schematisation, similarly to what is often done for city transport maps. In this case, we are providing a schematic overview of navigable waters, distorting intentionally the spatial correctness as in the London underground map. The map allows river boat travellers to view their options. The service chain involves the SDI nodes A, B, C and D. For each node, we specified a communication pattern similar to that of process expression 4. For example, the fundamental communication patterns of A and B are specified as:

$$\mathsf{A} \stackrel{\text{def}}{=} !\,\bar{c_1}r \,.\, c_4(s) \,.\, c_5(u) \,.\, c_6(v)$$
$$\mathsf{B} \stackrel{\text{def}}{=} !\,c_1(r) \,.\, \bar{c_2}s \,.\, c_3(u) \,.\, \tau \,.\, \bar{c_4}v$$

And services C and D being defined in similar style. The service orchestration in this scenario is typically of the form

$$\mathsf{A} \mid \mathsf{B} \mid \mathsf{C} \mid \mathsf{D}.$$

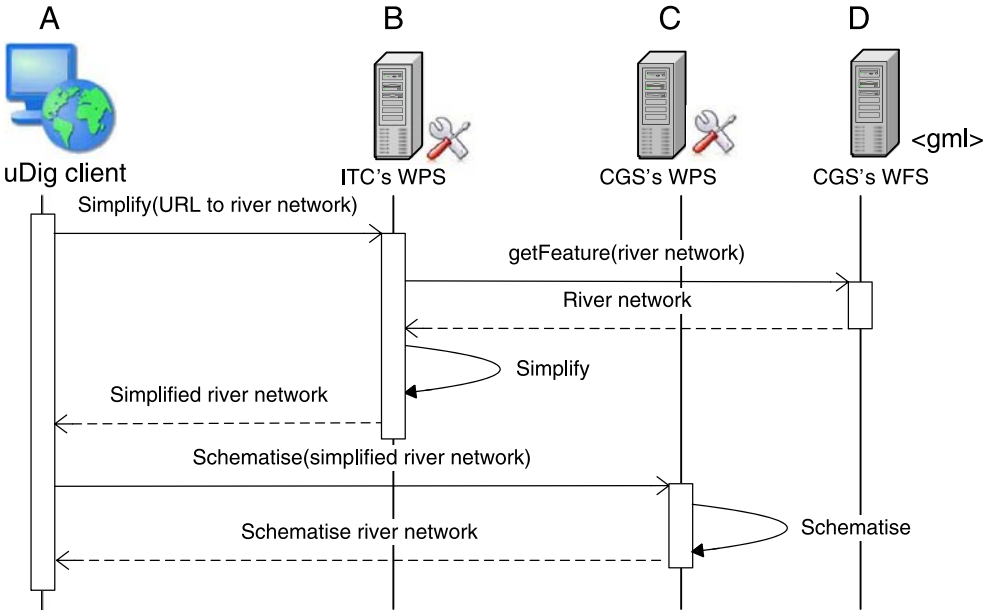

Fig. 3 Service chaining scenario for map schematisation (adapted from Hobona et al. (2009))

A metaserver of type *Workflow Server* will be responsible for such orchestration, for which it makes use of allowed workflow patterns in the SDI network. Additional metaserver functionality is the computation of metadata propagation in the service chain. Such metadata calculations will be restricted to proper states and dynamics as defined in Sections "Preliminaries" and "Proper states" (on specific metastates/dynamics).

Discussion and future work

Merging horizontal and vertical designs

Our discussion of horizontal and vertical designs has been intentionally abstract, for two reasons. First, space limitations prohibit the provision of full details on the design methods. We are currently developing a full methodical framework with a prominent role for a geospatially flavoured UML using OCL as the constraint language. Designs in that framework will be subjected to property-preserving transformations, allowing design alternatives, to arrive at trustworthy state-and-function systems. Similarly, work on process-algebraic designs for communication within SDI is underway.

A more important, second reason for our choice of abstraction lies in the required correspondence between functions from the one perspective, and services and communications from the other perspective. That correspondence appears natural at first, but is non-trivial when one gets to the details. Specifically, one needs to decide whether a service request-and-consumption is better specified as a single synchronous action in the SDI, or as a pair (request, response) of such actions. The methodological ramifications of either choice are unclear, and deserve further study.

Merging in the introspective design

Of specific interest in geospatial web service systems are the advanced metadata formats, and what can be done with them. Again, the spatial data and metadata deserves special attention as the aptness-for-use is generally a complex question, even more so than in regular web service systems (Schwinger et al. 2008).

It is an open issue how spatial web services are best characterised to allow their full exploitation. It is unlikely that the full semantics of the specification can ever be totally captured in metadata. The same holds for service chains, service orchestrations and services that offer service orchestrations. Here, one quickly reaches the realms, as well as boundaries, of the web beyond Web 2.0 (Raman 2009), for instance when introspection into the introspective capabilities is required.

Summary of the paper

This paper is about the 'design theory of SDI systems'. We believe that such a theory is not in existence, and that given the demanding requirements in SDI—full interoperability, advanced data types, advanced

functionality, deep semantics, and long lifetime expectancy—we must urgently make a start at constructing such a theory. It will hopefully allow us to build our SDI systems in more controlled ways, or rebuild them in those ways.

In the paper, we pass in review the various characteristics of a proper design theory. In brief, these are *completeness*, *abstraction*, *separation of concerns*, *repeatability*, *implementability*, *formal mathematical basis* and last but not least solid *grounding in the geoinformation domain*. Under these conditions, methods of transformational design can be developed, and we provide illustrations of these.

Why do we emphasise the need for a design theory in the SDI domain? Well, SDI systems are typically complex, and are being built to last. They will outlive the technology and standards that we use for their implementation today, and when we are going to 'upgrade' these systems later in their life cycle, it will be extremely useful to be able to fall back to mature design documents, in which the technology-of-the-day has been factored out through abstractions. The design of a complex system should, after all, be largely independent of the state-of-the-art technology.

But a design theory can serve other purposes than providing rationale for design and development steps of a system. It can also be used as a yardstick against which to hold already implemented systems, in retrospect, so as to understand more fully what the system is achieving, and what not. This is important because many SDIs are not at all the result of fundamental top-down design, and their history is rather more one of organismal growth and fine-tuning between its components.

Any design theory applies a generic paradigm for the artifacts that are its domain of study. We look at an SDI as a community of collaborating and communicating systems, each of which displays geoinformation needs and/or offerings. This gives us two important dimensions to SDI:

1. SDI as a *network of communication*, and
2. SDI as a *constellation of network nodes*.

Consequently,

The development of a design theory for SDI does not need to fall on infertile ground. After all, communication networks have been built before, and commonly with a design theory in the background. Likewise, web services-enabled information systems, which SDI nodes are, are also commonplace nowadays, and much of their design is based on appropriate formalisms.

Acknowledgements We are indebted to two anonymous reviewers of this paper, for their suggestions for improvements, and pointing out a number of inconsistencies in the original paper. We thank Fred Fonseca and Clodoveu Davis for their invitation to submit and their patience in handling our submission.

References

Annoni A, Friis-Christensen A, Lucchi R, Lutz M (2008) Requirements and challenges for building a European spatial information infrastructure: INSPIRE—Towards the spatial semantic web. In: van Oosterom P, Zlatanova S (eds) Creating Spatial Information Infrastructures, CRC Press, Taylor and Francis, pp 1–18

Arpinar IB, Sheth A, Ramakrishnan C, Usery EL, Azami M, Kwan MP (2006) Geospatial ontology development and semantic analytics. Trans GIS 10(4):551–575

Astrahan MM, Chamberlin DD (1975) Implementation of a structured English query language. Commun ACM 18(10): 580–588

Bergstra J, Klop J (1987) ACP: a universal axiom system for process specification. Cent Wiskd Inform 15:3–23

Bernard L, Kanellopoulos I, Annoni A, Smits P (2005) The European geoportal—one step towards the establishment of a European spatial data infrastructure. Comput Environ Urban 29(1):15–31

Brodersen L (2008) Geo-communication and information design. Forlaget Tankegang A/S

Carrara P, Fortunati L, Fresta G, Gomarasca M, Piazza BL, Poggioli D (2004) A methodological approach to the development of applications in a SDI environment. In: Proceedings of the 7th AGILE Conference on Geographic Information Science, Heraklion, Greece

Codd E (1970) A relational model of data for large shared data banks. Commun ACM 13(6):377–387

Craglia M, Goodchild MF, Annoni A, Câmara G, Gould M, Kuhn W, Mark D, Masser I, Maguire D, Liang S, Parsons E (2008) Next-generation Digital Earth: a position paper from the Vespucci initiative for the advancement of geographic information science. International Journal of Spatial Data Infrastructures Research 3:146–167

Crompvoets J, Bregt A (2003) World status of national spatial data clearinghouses. J Urban Reg Inf Syst Assoc 15(APA 1):43–50

Davis C, Fonseca F, Câmara G (2008) Understanding global change: the role of geographic information science in the integration of people and nature. In: Proceedings workshop SDI for the Amazon, position paper

de Man W (2006) Understanding SDI; complexity and institutionalization. Int J Geogr Inf Sci 20(3):329– 343

Feng Z, Yin J, Zhou J (2008) E-business processes composition based on pi-calculus technology. International Symposium on Information Science and Engineering 2:224–227

Georgiadou Y, Puri SK, Sahay S (2005) Towards a potential research agenda to guide the implementation of spatial data infrastructures—a case study from India. Int J Geogr Inf Sci 19(10):1113–1130

Hobona G, Jackson M, Gould M, Higgins C, Brauner J, Matheus A, Foerster T, Nash E, Lemmens R et al (2009) Establishing a persistent interoperability test-bed for European

geospatial research. In: Haunert J, Kieler B, Milde J (eds) Proceedings of the 12th Agile internationational conference: advances in GIScience, IKG, Leibniz University of Hanover, Hanover, Germany, p 10

Kroiß C, Koch N (2008) UWE metamodel and profile: user guide and reference. Tech. Rep. 0802, Institute for Informatics Ludwig-Maximilians-Universität München, München, Germany, version 1.0

Lance K (2003) Spatial data infrastructure in Africa. GIS Development July:6 pp

Lehto L (2007) Schema translations in a web service-based SDI. In: Proceedings of the 10^{th} AGILE international conference on geographic information science, Aalborg University, Denmark

Lucchi R, Mazzara M (2005) A *pi*-calculus based semantics for WS-BPEL. Journal of Logic and Algebraic Programming 70:96–118

Maguire DJ, Longley PA (2005) The emergence of geoportals and their role in spatial data infrastructures. Comput Environ Urban Syst 29:3–14

Masser I (1999) All shapes and sizes: the first generation of national spatial data infrastructures. Int J Geogr Inf Sci 13(1):67–84

Masser I (2005) GIS Worlds—creating spatial data infrastructures. ESRI, Redlands

Milner R, Parrow J, Walker D (1992) A calculus of mobile processes, part I/II. Journal of Information and Computation 100:1–77. http://www.lfcs.informatics.ed.ac.uk/reports/89/ECS-LFCS-89-85/

Mohammadi H, Rajabifard A, Binns A, Williamson IP (2006) Bridging SDI design gaps. Coordinates Magazine 2(5): 26–29

Open Geospatial Consortium Inc (2004) Geospatial portal reference architecture—a community guide to implementing standards-based geospatial portals, version 0.2 edn. OGC 04–039, OGC

Parrow J (2001) An introduction to the π-calculus. In: Bergstra JA, Ponse A, Smolka SA (eds) Handbook of Process Algebra, chap 8. Elsevier, Amsterdam, pp 479–543

Raman T (2009) Toward 2^{w}, beyond Web 2.0. Commun ACM 52(2):52–59

Schwinger W, Retschitzegger W, Schauerhuber A, Kappel G et al (2008) A survey on web modeling approaches for ubiquitous web applications. International Journal of Web Information Systems 4(3):234–305

Vieira PR, Lunardi OA, Correia AH, Issmael LS (2008) Spatial data infrastructure for Amazon. In: Proceedings workshop SDI for the Amazon, position paper

Wagner R (2006) A roaming-enabled SDI (rSDI): balancing interests, opportunities, investments and risks. In: Proceedings GSDI–9 conference, Santiago de Chile, Chile

Williamson I, Rajabifard A, Feeney MEF (eds) (2003) Developing spatial data infrastructures—from concept to reality. CRC, Boca Raton