A Unified Formalism for Complex Systems Architecture

Ph.D. in Computer Science

by

Boris Golden

Defended on May 13^{th} 2013 in front of:

M. Daniel KROB	École Polytechnique	Advisor
M. Marc AIGUIER	École Centrale Paris (ECP)	Co-Advisor
M. Frédéric BOULANGER	Supélec	Reporter
M. Sylvain PEYRONNET	Université de Caen	Reporter
M. Marc POUZET	École Normale Supérieure (ENS)	Reporter
M. Éric GOUBAULT	CEA	President
M. Patrice PERNY	Université Paris 6	Examiner

École Polytechnique - LIX

Chapter 1

Introduction

1.1 Complex industrial systems

Industrial systems are typically artificial objects designed by men, involving heterogeneous components (mostly: hardware, software, and human organizations) working together to perform a mission. In this thesis, we are interested in modeling the functional behavior of such systems, and their integration. From a practical point of view, our aim is to give a unified formal semantics to the concepts manipulated on a daily basis by engineers from various fields working together on the design of complex industrial systems. Indeed, they need formal tools to reason on & model those systems in a unified & consistent way, with a clear understanding of the underlying concepts.

We will model complex industrial systems as heterogeneous integrated systems, since our work highlights two aspects of the complexity of such systems:

- the *heterogeneity* of systems, that can be naturally modeled following continuous or discrete time, and that are exchanging data of different types[1]. Several specialized fields are involved in the design of a complex industrial system, making it difficult to keep a unified vision of this system and to manage its design.

- the *integration* of systems, i.e. the recursive mechanism to build a system through the synthesis of smaller systems working together, and whose behaviors will be described at a more concrete level (i.e. a finer grain). There are many interrelations between a possibly huge number of components, and there are recursive levels of integration.

The concept of *complex systems* has led to various definitions in numerous disciplines (biology, physics, engineering, mathematics, computer science, etc).

[1] Data encompasses here all kinds of elements that can be exchanged between real objects. We distinguish three kinds of homogeneous systems: hardware/physical systems (transforming continuous physical parameters), software systems (transforming and managing discrete data), and human/organizational systems (organized through processes).

One speaks for instance of dynamical, mechanical, Hamiltonian, hybrid, embedded, concurrent or distributed systems (cf. [6, 8, 45, 53, 60]). A minimalist informal definition consistent with (almost) all those of the literature is that a *system* is "a set of interconnected parts forming an integrated whole", and the adjective *complex* implies that a system has "properties that are not easily understandable from the properties of its parts". In the mathematical formalization of *complex systems*, there are today two major approaches: the first one is centered on understanding how very simple but numerous elementary components can lead to complex overall behaviors (e.g. cellular automatas), the second one (that will also be ours) is centered on giving a precise semantics to the notion of system and to the integration of systems to build greater overall systems.

When mathematically apprehended, the concept of system (in the sense of this second approach) is classically defined with models coming from:

- control theory and physics, that deal with systems as partial functions (dynamical systems may also be rewritten in this way), called transfer functions, of the form:

  $$\forall t \in T,\ y(t) = F(x, q, t)$$

  where x, q and y are inputs, states and outputs dataflows, and where T stands for time (usually considered in these approaches as continuous (see [60, 5, 21]).

- theoretical computer sciences and software engineering, with systems that can be depicted by automaton-oriented formalisms equivalent to timed transition systems with input and output, evolving on discrete times generally considered as a universal predefined sequence of steps (see for instance [37, 10, 35, 26, 19, 18]). There is also a purely logical approach for representing discrete abstract systems. The core modelling language in this direction is probably Lustre [35, 20]. A reasoned overview of all these approaches can be found in [39].

However all these models do not easily allow to handle systems with heterogeneous time scales. The introduction of a more evolved notion of time within models involves many difficulties, mainly the proper definition of sequential transitions or the synchronization of different systems exchanging dataflows without synchronization of their time scales. Dealing with an advanced definition of time will typically imply to introduce infinity and infinitesimal (for instance with non-standard real numbers).

To address this challenge, the theory of hybrid systems was developed jointly in control theory (see [60, 66]) and in computer science (see [36, 6, 41]). A serious issue with this theory is however that the underlying formalism has some troubling properties such as the Zeno's paradox which corresponds to the fact that an hybrid system can change of state an infinite number of times within a finite time with the convergence of a series of decreasing durations

which should be avoided in a robust modeling approach. Other interesting and slightly different attempts in the same direction can also be found in Rabinovitch and Trakhtenbrot (see [52, 61]) who tried to reconstruct a finite automata theory on the basis of a real time framework, or in [67]. Hybrid automatas (see [36]) are another classical model for representing abstract hybrid systems.

Moreover, none of these models address the integrative & architectural dimensions of complex industrial systems (an approach similar to ours on the structure of complex systems, but without the introduction of heterogeneous time, has been carried out in [3], using a coalgebraic formalism). There is therefore a great challenge on being able to unify in a same formal framework mathematical methods dealing with the definition & design of both continuous and discrete systems, and at the same time being able to define the integration & architecture of such systems in the same formalism. This will be at the center of our approach.

1.2 Systems Engineering

When dealing with complex industrial systems containing heterogeneous components, engineers face problems with the semantics of the models they work with to describe real systems when they involve a large number of heterogeneous elementary systems.

To build modern industrial systems, it has thus been necessary to create complex engineering models & processes being able to deal with a huge number of engineers coming from many different domains. Indeed, because of the size of such modern industrial systems (trains, planes, space shuttles, etc), their realization leads to major conceptual and technical difficulties. The reason is that it is difficult, or even impossible, for one single person to completely comprehend such systems globally. Although this designation is not precise nor universal, it is naturally associated with industrial systems whose design, industrialization and change lead to important and difficult problems of integration, directly related to both the huge number of basic components integrated at multiple levels, and the important scientific and technological heterogeneity of such systems (generally involving software, hardware/physical and human/organizational parts).
To manage the complexity of such systems, some methods have emerged during the last century. We can trace the origin of these methods to the Cold War where USA had to build defence systems for which the issues of data management, decisions and military riposte had been taken into account as a whole, in an integrated and consistent way, in order to ensure a short reaction time between a Soviet attack and an American counter-attack. From there, a body of knowledge called *Systems Engineering*, focused on the integration mastery of large industrial systems has progressively emerged since the 50's. Hence, Systems Engineering consists of a set of concepts, methods and good organizational and technical practices that the industry had to develop to be able to deal with the complexity of industrial systems (see [11, 42, 57, 62] for more details on this subject).

But in fact, Systems Engineering is "just" the application to engineering of a more general thought paradigm, called *systems approach* (also often referred to as *systems thinking*). Systems approach focuses on interactions between systems, and views such systems as black boxes described only through their functional behavior and their internal state. In systems approach, a system is thus a black box receiving and emitting flows, and characterized by an internal state. A system can itself be decomposed into a set of interconnected subsystems. It is therefore an observational modeling of systems. Systems approach also implies to step back with a high-level point of view seeing any system as being part of an overall greater system (i.e. a system is in interaction with other systems in its environment).

A key assumption of systems approach is thus to consider that any system is only in interaction with other systems, and that the behavior of any real system[2] can be explained within this framework. The main advantage of this approach is that it helps to understand how things influence one another within a whole (with often unexpected long-term or long-distance influences). All interactions between systems are captured by logical flows, figuring a unidirectional transmission of elements (which can be material, energetic or informational). "Logical" here means that it only models the exchange of data between elements, and not the way this exchange really occurs. For instance, the physical reality behind a flow between two real systems (like the delay or the rate of transmission of the network cable between two computers) will itself be modeled by a specific type of system called *interface* (it would here take into account the physical properties of the cable, when the flow would only account for the logical exchange of data).

The systems approach is a very powerful tool to model a lot of real-life objects and situations. We give a simple example to illustrate the main concepts of systems approach. Imagine an individual, John, sitting in front of its computer and using it. John has on its table a bottle of water and a glass he uses when he is thirsty. We want to model the overall real system[3] (which is here closed, i.e. not exchanging data with its environment). We should characterize the following objects at the right abstraction level:

- *systems*: those are all the objects involved. Here: glass, bottle, computer, John (the table will not be useful in our modeling).

- *states*: each system has a set of possible states. E.g. the glass can be 'not_empty' or 'empty'.

[2] In the literature, the real object and its model are often confused and both called *system*. When clarification is needed, we will call *real system* any object of the real world which behavior we want to explain as a transformation of flows of data. We will call *system* the mathematical object introduced to model real systems.

[3] As every model, it is not exhaustive since focused on given aspects of the real system considered to meet the (implicit) goals of the modeling.

- *data*: each system can receive or send data. E.g. the glass can send water, or a visual stimulus (indicating the level of water in the glass).
- *flows*: each system has inputs and outputs that allow it to exchange data with other systems. E.g. the bottle can send water to the glass (when the bottle is not empty). But the glass may send water to the computer, even if it is not its intended use.
- *behaviors* (functions and states): each system will have specific behaviors that will make it generate outputs and change its states, according to time and inputs. E.g. when the glass is empty and receives water, it becomes not_empty. But if it receives a drinking move, it sends water to John and becomes empty.
- *time* of the overall system: a time scale of description has to be chosen to be consistent with the model. E.g. a step of 1 minute for the time scale can be a consistent choice here.

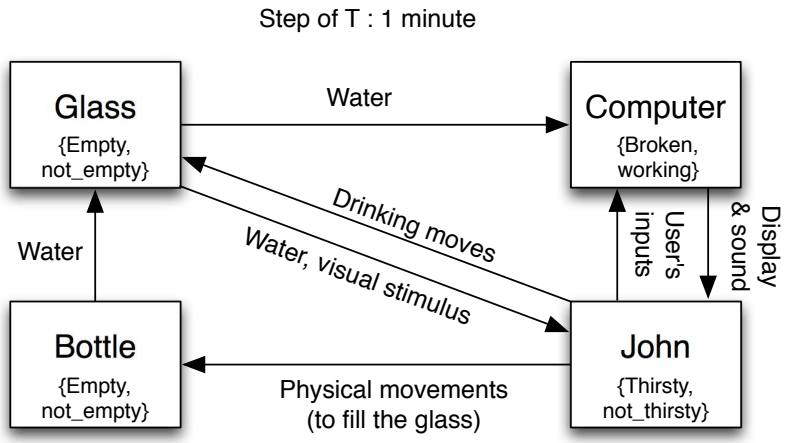


Figure 1.1: Example of an elementary systemic modeling

All the concepts introduced here are informal. We will give a semantics to all the objects of this intuitive "graphical" language used in systems approach.

1.3 What is systems architecture?

1.3.1 A definition

Systems Architecture is a generic discipline to handle systems (existing or to be created), in a way that supports reasoning about the structural properties of these objects.

Depending on the context, Systems Architecture can in fact refer to:

- the architecture of a system, i.e. a *model* to describe/analyze a system
- architecting a system, i.e. a *method* to define the architecture of a system
- a body of knowledge for "architecting" systems while meeting business needs, i.e. a *discipline* to master systems design.

At this point, we can only say that the "architecture of a system" is (similarly to the one of a building) a global model of a real system consisting of:

- a structure
- properties (of various elements involved)
- relationships (between various elements)
- behaviors & dynamics
- multiple views of elements (complementary and consistent).

We will not describe here the numerous issues raised (at every level of a company: corporate strategy, marketing, product definition, engineering, manufacturing, operations, support, maintenance, etc) by the design and management of complex industrial systems. But these issues can be summarized as:

- going from local to global, i.e. mastering integration and emergence
- building an invariable architecture in a moving environment.

In this context, Systems Architecture is a response to the conceptual and practical difficulties of the description and the design of complex industrial systems. Systems Architecture helps to describe consistently and design efficiently complex systems such as:

- an industrial system (the original meaning of Systems Architecture)
- an IT infrastructure (Enterprise Architecture)
- an organization (Organizational Architecture)
- a business (Business Architecture).

Systems Architecture will often rely on a tool called an *architecture framework*, i.e. a reference model to organize the various elements of the architecture of a system into complementary and consistent predefined views allowing to cover all the scope of Systems Architecture. Famous architecture frameworks are for instance: DoDAF, MoDAF or AGATE[4].

Finally, Systems Architecture will consider any system with a socio-technical approach (even when dealing with a "purely" technical system). In particular, during the design (or transformation) of a system, the systems in the scope of this design (or transformation) can be divided in two separated systems in interaction:

[4] A good overview of these frameworks can be found on Wikipedia: http://en.wikipedia.org/wiki/Enterprise_Architecture_framework

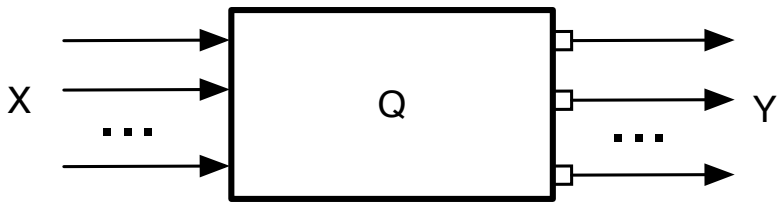
- the *product*, i.e. the system being designed or transformed
- the *project*, i.e. the socio-technical system (teams, tools, other resources and their organization following strategies & methods) in charge of the design or transformation of the product.

1.3.2 Fundamental principles

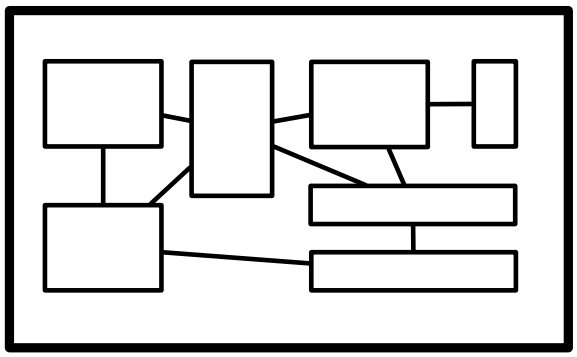
Whatever the type of system and the acception considered (model, method or discipline), Systems Architecture is based on 9 fundamental principles:

Thinking with a systemic approach

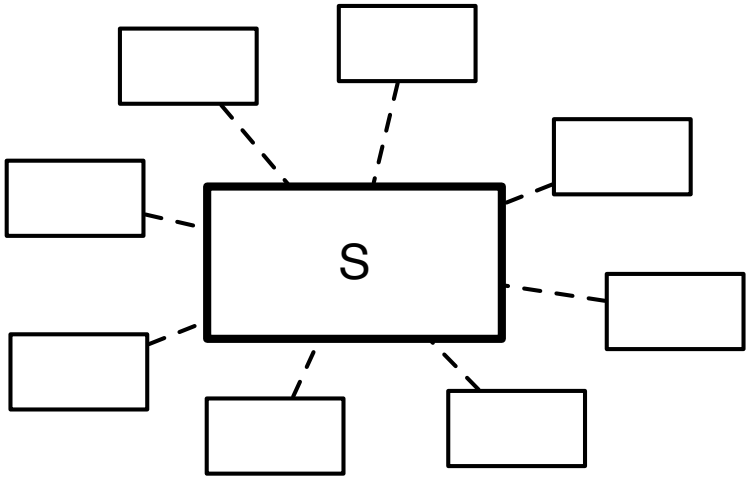
1. the objects of the reality are modeled as systems (i.e. a box performing a function and defined by its perimeter, inputs, outputs and an internal state). *Ex: a mobile phone is a system which takes in input a voice & keystrokes and outputs voices & displays. Moreover, it can be on, off or in standby. Overall, the phone allows to make phone calls (among other functions).*



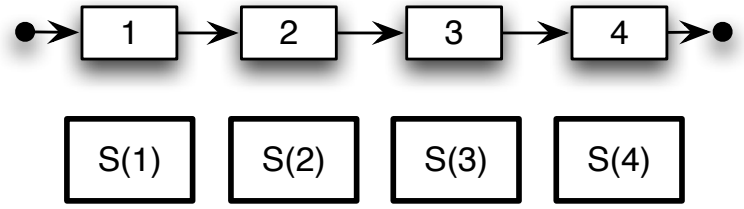
2. a system can be broken down into a set of smaller subsystems, which is less than the whole system (because of emergence). *Ex: a mobile phone is in fact a screen, a keyboard, a body, a microphone, a speaker, and electronics. But the phone is the integration of all those elements and cannot be understood completely from this set of separate elements.*



3. a system must be considered in interaction with other systems, i.e. its environment. *Ex: a mobile phone is in interaction with users, relays (to transmit the signal), repairers (when broken), the ground (when falling), etc. All these systems constitute its environment and shall be considered during its design.*

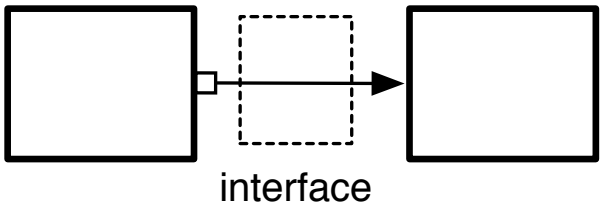


4. a system must be considered through its whole lifecycle. *Ex: a mobile phone will be designed, prototyped, tested, approved, manufactured, distributed, sold, used, repaired, and finally eventually recycled. All these steps are important (and not only the moment when it is used).*

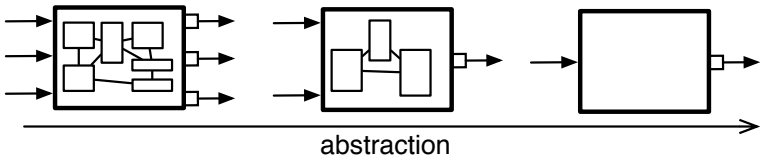


Reasoning according to an architecture paradigm

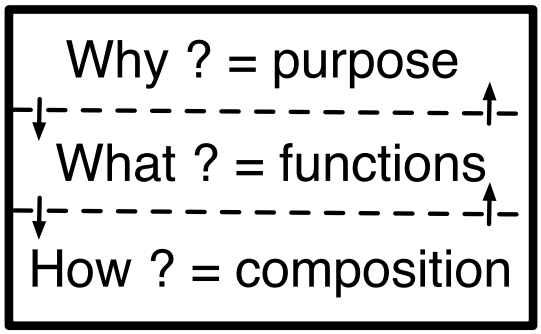
5. a system can be linked to another through an interface, which will model (when needed) the properties of the way they are linked in the reality. *Ex: when phoning, our ear is in direct contact with the phone, and there is therefore a link between the two systems (the ear and the phone). However, there is a hidden interface: the air! The properties of the air may influence the link between the ear and the phone (imagine for example if there is a lot of outside noise).*



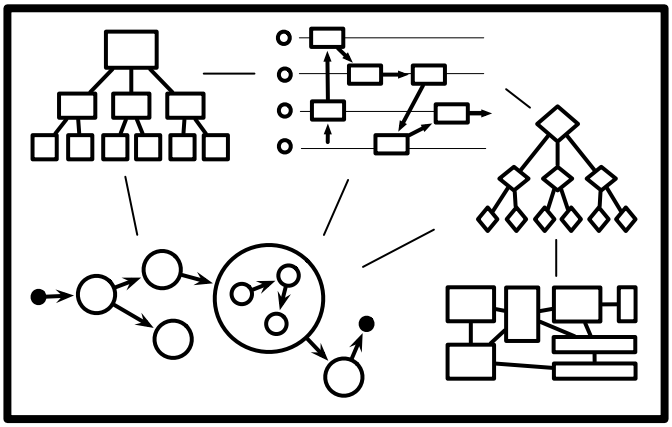
6. a system can be considered at various abstraction levels, allowing to consider only relevant properties and behaviors. *Ex: do you consider your phone as a device to make phone calls (and other functions of modern phones), a set of material and electronics components manufactured together, or a huge set of atoms ? All these visions are realistic, but they are just at different abstraction levels, whose relevancy will depend on the purpose of the modeling.*



7. a system can be viewed according to several layers (typically three at least: its purpose, its functions, and its construction). *Ex: a phone is an object whose purpose is to accomplish several missions for its environment: making phone calls, being a fashionable object, offering various features of personal digital assistants, etc. But it is also a set of functions organized to accomplish these missions (displaying on the screen, transmitting signal, delivering power supply, looking for user inputs, making noise if necessary, etc). And finally, all these functions are implemented through physical components organized to perform these functions.*



8. a system can be described through interrelated models with given semantics (properties, structure, states, behaviors, datas, etc). *Ex: from the point of view of properties, the phone is a device expected to meet requirements like "a phone must resist to falls from a height of one meter". But a phone will also change state: when a phone is off and that the power button is pressed, the phone shall turn on. Function dynamics of the phone are also relevant: when receiving a call, the screen will display the name and the speaker will buzz, but if the user presses no button the phone will stop after 30 seconds... This will typically be described with diagrams in modeling languages like UML or SysML.*



9. a system can be described through different viewpoints corresponding to various actors concerned by the system. *Ex: marketers, designers, engineers (in charge of software, electronics, acoustics, materials, etc), users, sales, repairers... All these people will have different visions of the phone. When the designer will see the phone as an easy-to-use object centered on the user, the engineer will see it as a technological device which has to be efficient and robust. A marketer may rather see it as a product which must meet clients' needs and market trends to be sold. All these visions are important and define the system in multiple and complementary ways.*