Modeling IT Systems in TRIZ

Jerzy Chrząszcz, Tiziana Bertoncelli, Barbara Gronauer, Oliver Mayer, Horst T. Nähler

Modeling IT systems in TRIZ

Jerzy Chrząszcz[1] [0000-0003-3257-9726], Tiziana Bertoncelli[2],
Barbara Gronauer[3] [0000-0001-7054-321X], Oliver Mayer[4], Horst Nähler[5] [0000-0002-8853-815X]

[1] Warsaw University of Technology, Institute of Computer Science, 00-665 Warsaw, Poland
[2] ANSYS Germany GmbH, 83624 Otterfing, Germany
[3] StrategieInnovation, 36088 Huenfeld, Germany
[4] Bayern Innovativ, 90402 Nürnberg, Germany
[5] c4pi - Center for Product Innovation, 36088 Huenfeld, Germany

jerzy.chrzaszcz@pw.edu.pl

Abstract. Today's world extensively uses Information Technology (IT) and heavily depends on IT systems. The evident and justified need to model IT systems in TRIZ projects faces some doubts or objections because the fundamental concepts of classic TRIZ were formulated before computer times. This paper aims to analyze terminology and guidelines regarding Function Analysis for products and Flow Analysis in the context of modeling IT systems and devising necessary adjustments so that components like hardware, software, and data may be modeled consistently.

We describe the approach taken and introduce a minimal complete IT system using the TRIZ approach, with a detailed description of components and functions supported by examples of IT systems. A conceptual perspective allowing for uniform modeling of mechanical and IT systems is intended to alleviate the problems of trainers and practitioners encountered in this area and increase TRIZ acceptance in the IT industry and IT communities.

Keywords: TRIZ, Function Analysis, IT systems, Hardware, Software, Data, Flow Analysis, Information Technology Modeling, IT Modeling.

1 Introduction

The proliferation of Information Technology (IT) and extensive use of various IT solutions in countless application areas call for a systematic approach to modeling IT systems in TRIZ. The IT systems, as we know them today, are usually described as consisting of hardware (computers) processing data under the control of software (programs). The main difference between these building blocks is that programs and data are intangible entities.

This property is perceived as the key success factor of the computer systems, as both programs and data may be easily modified and quickly transmitted to any location. On the other hand, the intangibility raises doubts and objections in the TRIZ community because the foundations of the methodology were created before computer times.

In particular, the basic definitions used in Function Analysis and Substance-Field Analysis are misleading, at best, in this respect. The definitions of *Function*, *Component*, and *Parameter* taken from selected reputable sources are given in Tab. 1 below. As can be seen, some items are missing, and some differ significantly in scope or approach. The definitions coming from the MATRIZ-approved sources seem the most compatible with each other, but still [1,2] require the objects to be *material*, while in [3], this adjective does not appear. Although there are also some differences between definitions of *Substance* and *Field*, the sources generally agree that a substance does have and a field does not have a rest mass.

Table 1. Selected definitions of the basic terms used in Function Analysis.
The original capitalization of the terms has been removed for easier comparison.

Source	Function	Component	Parameter
Glossary of TRIZ and TRIZ-related terms [1]	specification of an action performed by a material object (function carrier) that results in a change or preservation of a value of an attribute of another material object (object of the function).	a material object (substance, field, or substance-field combination) that constitutes a part of a technical system or its supersystem; a component might represent both a single object and a group of objects.	a variable dimensional or dimensionless measurable factor, either specific or aggregated, that participates in the definition of an attribute of a material object of a technical system or its supersystem and determining its borders and behaviors. (…)
Selected Topics for Level 1 Training [2]	an action performed by one material object (function carrier) to change or maintain a parameter of another material object (object of the function).	a material object (substance, field, or substance-field combination) that constitutes a part of the engineering system or supersystem.	a comparable value of an attribute.
Level 3 training materials [3]	an action performed by one component (function carrier) to change or maintain a parameter of another component (object of function)	an object (substance, field or substance-field combination) that constitutes a part of an engineering system or supersystem	a comparable value of an attribute
A brief glossary of basic concepts and terms of TRIZ [4]	a change, stabilization or measurement of certain parameters of an object of function (or product) by the influence of the carrier of the function (tool) on it. (...)	part of the system as an element (substance) or as a field	a quantity that characterizes a property of a process, phenomenon or system; the parameters indicate how this system (process) is different from others (...).
VDI 4521 Part 1 [5]	effect from a system or a system component upon others which changes, eliminates, or maintains a parameter of the other component or system	n/a	n/a

A rest mass neatly corresponds with the material nature of a substance, but how a field may be perceived as a material object when it does not have a rest mass and contains no matter? Moreover, if we attempt to identify the components of an IT system, it seems reasonable to perceive hardware as a substance and software and data – as fields. Therefore, if one insists that a field must be a material object, the conclusion seems obvious: neither software nor data may be a legitimate system component (and such a statement was communicated during some TRIZ workshops, usually without hardcopy on slides).

The first viable solution to this puzzle is to remove the *material* adjective from the definitions of the component and function, just like it was done in [3], eliminating the doubts regarding a material field. The second way is less obvious, as it requires reconsidering the native meaning of this adjective. Although it is not emphasized in international literature, TRIZ originated from the materialistic and dialectic worldview [6]. Therefore, the adjective *material* in the mentioned definitions should be interpreted as *objective*, *measurable*, and *capable of interacting with other entities* rather than *made of matter*. This is an example of a notion that was literally "lost in translation" during the internationalization of TRIZ methodology. Since software and data are undoubtedly material in a sense mentioned above, we will consider them legitimate field-type components of technical systems in the following sections.

Several authors have already addressed the topic of applying TRIZ in the IT domain. A comprehensive overview of software-related works is given in [7], which also provides some forecasts for this area. Other interesting publications focus on functional modeling [8,9], Substance-Field modeling [10], transferring the 40 Inventive Principles to the IT domain [11-14], and Laws of Evolution of information systems [15]. Nevertheless, the appropriateness of TRIZ for IT still raises doubts – see two quotes from [9]: *"Key Assumptions: Components in Function Model can be material objects only (…) Uncertainties: Uncertainty how to deal with functions in IT systems"*.

2 The building blocks

The first decomposition of a computer system is between hardware and software. The hardware comprises processor(s), memory, and input/output devices (i/o) required for communication with the rest of the world – in particular, sensors and effectors. There are some subcategories, as memory may appear in several layers of a computer system (processor registers, cache memory, operating memory, etc.) and several technologies (semiconductor, magnetic, optical, ferroelectric, etc.) accounting for specific characteristics of the devices – for instance, contents volatility vs. non-volatility upon power off.

Software is generally meant as programs, being sequences of instructions for the hardware units, but there are also some shades here. For instance, the system start-up (boot-up) program is typically stored in non-volatile memory, and such built-in programs are often called firmware to differentiate it from software applications, which are usually loaded from a storage device into operating memory to be executed. To configure the i/o devices, several settings must be preset by writing specific values to the internal registers of i/o controllers, so these values must also be stored in the non-volatile memory.

The data also appears in two main categories, namely the constants and the variables. As the names indicate, the difference is that variables' values may be modified during processing while the constants retain their values. Some constants are usually built in the program code, which may be stored in read-only memory. Hence, such constants should be considered as belonging to the program rather than data. Read-only memory is also used for storing data items having fixed values, such as predefined lists (e.g. month names) or error messages. The variables, on the contrary, must be stored in read-write memory, and they are used for keeping the values acquired from sensors, auxiliary data structures, intermediate results, and final results.

The memory stores programs and data for different purposes. Programs are intended to be executed by the processor, while data is intended to be manipulated (processed) by programs. It should be noted that at the physical level, memory stores just binary patterns so that data and instructions cannot be directly differentiated from each other. As a consequence, instructions stored in rewritable memory may be manipulated as data (which allows for creating self-modifying programs), and also data may be executed as instructions (by omission or as a security breach).

The processor fetches and executes instructions from subsequent memory cells unless a control instruction (e.g. jump) forces a non-sequential transition. The instruction execution cycle comprises (1) fetching instruction, (2) decoding instruction, (3) reading operands from the source locations, (4) executing respective operation, e.g. addition, and (5) writing operation result into the destination location. This cycle covers the most complex scenario, as some instructions do not need operands, some do not produce a result, etc. The processing capabilities are determined by the processor's instruction set (defining possible data manipulations) and the addressing modes (defining possible ways of accessing data items during processing).

3 Function Analysis for products

As was mentioned before, at a high level of abstraction, a computer system is usually described as a combination of hardware and software, so let us build a function model of such a system. We should start with defining the main function and boundary of the system, and the first ambiguity appears here, as the two generic types of processing systems differ significantly.

An embedded processing system (industrial controller, signal converter, etc.) may work without direct interaction with the user, and so its main function may be described as *to process data*. An interactive processing system, in turn (a PC, smartphone, etc.), is controlled directly by the user and provides results to the user so that the main functions may be described as *to process data* and *to inform user*. Moreover, taking into account that computer systems may interact with other computer systems instead of (or in addition to) humans, we may generalize *user* as *operator*.

Another source of ambiguity is that equivalent data processing may be implemented in two distinct ways. Either dedicated, application-specific hardware (e.g. ASIC) or generic software-controlled hardware (programmable processor) may be used. In the former variant, we have only hardware. The sequence of operations is defined with the

preconfigured interconnections, and the operations are determined by the functions of the execution units embedded in the structure. The latter variant comprises hardware with all necessary execution units used on-demand and software determining the sequence of operations. We will only address hardware-software systems here.

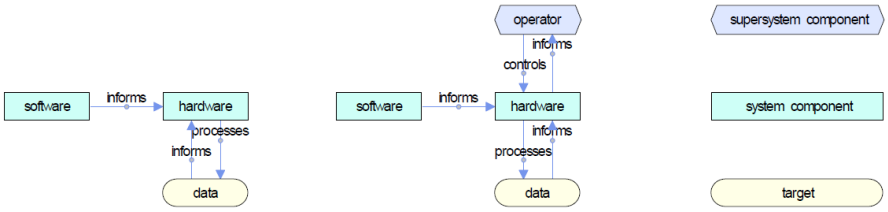

Fig. 1. Basic IT system models: embedded (left) and interactive (right).

For variant 1, the system comprises hardware (substance) and software (field), while the data (field) is a supersystem component and the functions are as follows:

– *software informs hardware* (reflects fetching instructions by the processor),
– *data informs hardware* (reflects reading values of data items),
– *hardware processes data* (reflects modifying and writing data items).

For variant 2, the system comprises hardware (substance) and software (field), while the data (field) and the operator (substance) are supersystem components. The functions *software informs hardware*, *data informs hardware*, and *hardware processes data* are same as above, and the new functions are:

– *operator controls hardware* (as input from the operator goes via the input devices),
– *hardware informs operator* (reflects the operation of the output devices).

Because variant 1 is a subset of variant 2, we will only consider modifications of the latter. The first extensions indicate the *parameters* as a system component (field) and distinguish *input data* (operands) from *output data* (results), as shown in Fig. 2. Parameters are meant as configuration settings e.g. coefficients in a weighted sum calculation.

– *parameters informs hardware* (reflect reading parameter values),
– *hardware processes parameters* (modifying and writing parameter values),
– *input data informs hardware* (same as *data informs hardware*),
– *hardware generates output data* (reflects modifying and writing result values).

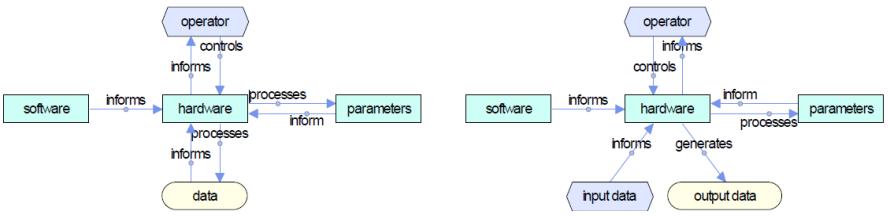

Fig. 2. A model of an IT system with indicated parameters. Data may be considered as a unified input and output component (left), or separate input and output data components (right).

A model may also be given finer granularity, like in Fig. 3, where *hardware* is decomposed into *processor*, *memory*, and *i/o devices* (substances), and memory contents are decomposed into *programs*, *parameters*, and *variables* (fields). The functions regarding supersystem components differ from the previous model to indicate that all interactions between the processor and the outside world go through the i/o devices. The internal functions are as follows:

- *processor controls i/o devices* (reflects indicating operations to be performed by i/o devices and, as a second function, writing data to i/o ports),
- *i/o devices inform processor* (reflects reading data from i/o ports),
- *processor controls memory* (reflects indicating operations to be performed by memory and, as a second function, writing contents to memory locations),
- *memory informs processor* (reflects reading memory contents, including program instructions, parameter values, and variable values),
- *memory holds program / parameters / variables* (reflect that memory cells are kept unchanged between write cycles).

Two additional functions are indicated in the right diagram to model the capability of direct data transfers between memory and i/o devices without consuming processor time, known in IT as Direct Memory Access functionality (*i/o devices control memory*, and *memory informs i/o devices*).

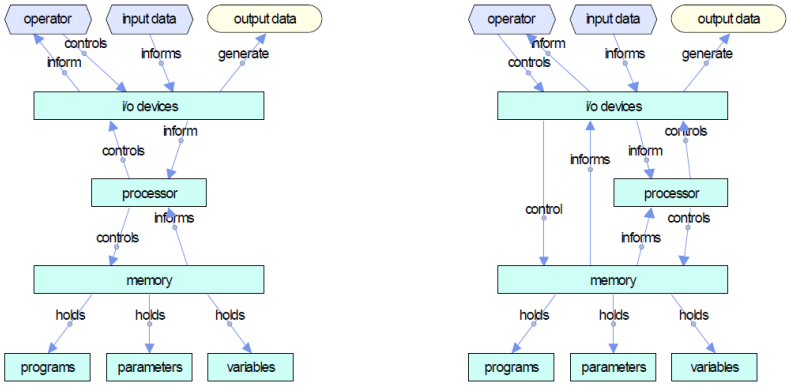

Fig. 3. An expanded model of an IT system with i/o data transfers provided by processor (left) and with Direct Memory Access capabilities (right).

4 A complete IT system

The concept of a complete technical system, related to the Trend of Increasing System Completeness, works well in TRIZ for mechanical systems. Surprisingly, respective definitions also vary between the sources, as shown in Tab.2. The original Altshuller's description in [16] indicates four mandatory components, namely *engine*, *transmission*, *control*, and *tool*.

The same composition is indicated in [1] and in newer publications, such as [17], where the *energy source* is indicated in the supersystem. Other publications [2,3,18] show the energy source as a mandatory component of a complete system instead of the engine, and some even refer to the engine as an example of the energy source.

Yet another version appears in [5], where energy source and object (product) are indicated as system components in addition to the four mentioned in the original definition. Finally, a generalized composition was devised in [19], with the *source of matter, energy, and information* (located on the system boundary – i.e. possibly or partially in the supersystem), supposedly replacing the energy source and engine, and a *converter* (instead of transmission).

Table 2. Selected definitions referring to the complete technical system.
The original capitalization of the terms has been removed for easier comparison.

Source	Definition
Glossary of TRIZ and TRIZ-related terms [1]	A technical system that according to the trend of technical system completeness, includes at least four components (subsystems) which provide functions of engine, transmission, control unit, and working unit.
Selected Topics for Level 1 Training [2]	Control system: a common functional part of the engineering system that controls how the other parts function. For example, a thermostat in an air-conditioning system. Energy source: a common functional part of the engineering system that generates energy to operate the system. For example, an engine in a car. Transmission: a common functional part of an engineering system and its supersystem that transfers a field (energy) from an energy source to an operational device. Operating tool: a typical function part of an engineering system that usually performs the most important basic functions.
Level 3 training materials [3]	As an engineering system evolves, it acquires the following typical functions: the function of operating agent, the function of transmission, the function of energy source, the function of control system.
A brief glossary of basic concepts and terms of TRIZ [4]	n/a
VDI 4521 Part 1 [5]	System consisting of the elements: energy source, drive, transmission, working means, control, object and, if applicable, further interfaces to the supersystem.

The proposed diagram of a complete IT system (see Fig. 4) has been derived from Altshuller's version. The software is perceived like fuel for the hardware engine, and this energy source may be located inside or outside the system. Data is the object of the processing, and the elementary operations indicated as the tool are determined by the instruction set of the processor. Control is provided by an operator (human or machine), and the instruction cycle is the transmission counterpart as this is how the "potential energy" stored in the software (after transformation into "kinetic energy" by hardware) is applied to data items in the form of specific instructions executed in a specific sequence.

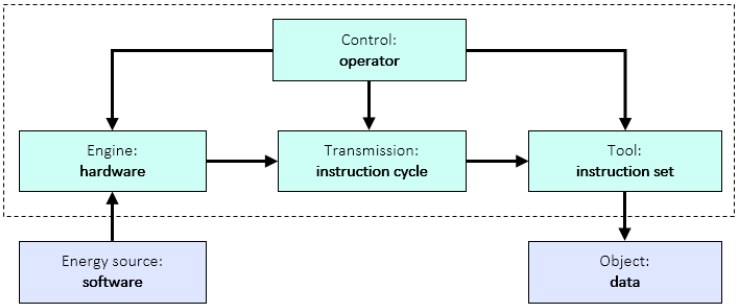

Fig. 4. The proposed composition of a complete IT system.

As mentioned in section 2, the instruction cycle is hardware activity supporting program execution by reading instructions and operands, performing required operations, and storing the results. The actual processing capabilities of a given processor depend on its instruction set. For example, some processors perform multiplication in a single clock cycle, others use a sequence of low-level operations, making multiplication instructions execute considerably longer than other instructions, while many processors do not have multiplication instructions at all. They still may multiply numbers, but this must be implemented in software. That is why the instruction cycle is proposed as the counterpart of transmission in a mechanical system, and the instruction set is considered a counterpart of a tool, being the component operating on the object directly. A similar diagram regarding the composition of a complete software system is presented in [20].

5 Flow Analysis

An IT system can as well be modeled using the Flow Analysis approach to describe the data manipulation process. In TRIZ, a distinction is made for the forms of flows:

- The *material flow* is about moving material objects with a mass, or around a large number of similarly movable objects, for example, cars, bottles, on a conveyor belt, etc. Such flows are usually neglected in IT systems unless we are interested in e.g. dust transfer that may affect cooling capabilities and lead to a system failure.
- The *energy flow* is about moving nonmaterial objects, like energy, acoustic, or light (photons). High-performance IT systems consume lots of energy, while portable systems, especially those providing critical functions, need sophisticated power management so that analyzing flows of energy is of great importance in IT systems.
- And third, explicitly, the *information flow* consisting of bits and bytes, broadcasting information, and the like is stated. Here is the explicit thought to run a nonmaterial object through a process and to drive and control this process. This type of flow is the basic area of interest when analyzing IT systems, as both software (instructions) and data must be transferred between components, and the overall performance of the system is affected by stalls, delays, insufficient throughput, and other deficiencies regarding the flow of information.

In the flow analysis, there are four categories of flows:

- A useful flow is where a material object, energy, or information is performing a useful function or is subject to a useful function.
- A harmful flow is where a material object, energy, or information is performing a harmful function or is subject to a harmful function.
- A wasted flow is where a material object, energy, or information is characterized by the loss of substance, energy, or information.
- A neutral flow where a material object, energy, or information is characterized by an irrelevant or insignificant effect on the technical system.

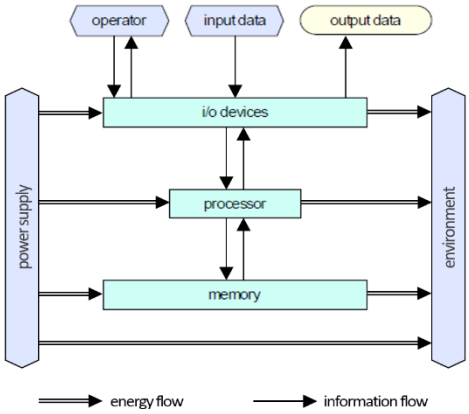

Fig. 5. Typical flows in a generic IT system.

A simplified diagram of energy and information flows in a properly operating IT system is shown in Fig. 5. Electricity from the power supply goes to the system building blocks indicated as single components (processor, memory, and i/o devices), although each of them may appear as several units, e.g. processor cores or memory modules. The mapping of the logical components onto physical devices may be much more complicated, e.g. latest generations of processors have cache memories, memory controllers, and sometimes also graphic controllers integrated on-chip.

Heat dissipation in the system is shown as energy flows sinking to the environment, which should be considered harmful. Thermal interactions between system components are neglected in the diagram for simplicity, while in reality, some elements may transfer heat to others. Finally, all information flows are shown as useful and adequate, while in a real system, there may be some bottlenecks or other flow disadvantages affecting system performance, and the flow model is capable of reflecting such situations. On the other hand, the model may also represent parasitic flows – e.g. data breaches.

As in the previous diagrams, we ended the description of the situation by generating the output data, which is typical for batch processing. In control systems, on the contrary, some input data comes from the supervised object or process through sensors, and some output data goes to the operator or to effectors to control system operation in real-time. In such cases, some additional flows should be added in the supersystem.

6 Examples

In this paragraph, an example of using the FA for products for software applications is given; the system of choice is the so-called Corona-Warn-App [21], offered and maintained by the German Authorities – in particular Robert Koch-Institut (RKI) – for disease spreading control via contact tracing, introduced during the Covid-19 Pandemic. The example is meant to be the first tentative to apply TRIZ to decentralized software systems, which is also interesting from the data security perspective. In fact, decentralization was the key technology that was claimed to assure anonymity and data privacy protection during the app roll-out. Since the system is a software application for smartphones intended to work by gathering and processing data from millions of users at the same time, the chosen template for the function model is the one depicted in Fig. 2, that is, with separate input and output components, of type interactive. With respect to the specific application example, the different components are mapped as follows:

- operator: smartphone user (only two users are shown in the example for clarity),
- hardware: smartphone + Bluetooth Low Energy module,
- software: Robert Koch-Institut Corona-Warn-App (CWA),
- input data: rolling proximity ID codes (automatically encrypted); optional: infection status, test status, vaccination status of other users,
- output data: own rolling proximity ID codes (automatically encrypted); optional: infection status, test status, vaccination status,
- parameters: time of contact, distance of contact (signal damping), time since contact, transmission risk level of the contact (evaluated on the basis of infection begin and symptom status) with respect to other smartphones with active Corona-Warn-App.

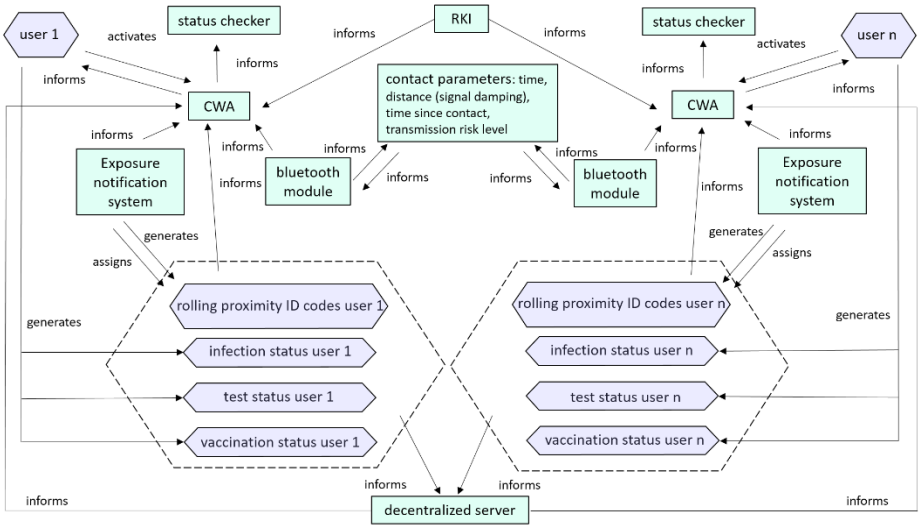

Fig. 6. Function model of a distributed IT system.

It can be observed that in this proposed function model, no function ranking has been performed. Moreover, only useful functions have been listed, while in order to identify bottlenecks and/or problematic system parts also, harmful functions should be identified. Nevertheless, for this model and for the software/IT function model in general, it is advisable to rank if the useful functions are performed inefficiently or if they may represent a "weak link", i.e. if there is a danger of data loss.

The qualification of the functions (useful, performed sufficiently, insufficiently or excessively, harmful) may also vary according to the specific problem statement: for this example, the main function at the time of the App deployment was *to inform user* through a risk notification that was tuned to the disease understanding and level of danger to be performed satisfactorily. This changed during the pandemic evolution and resulted in being excessively performed at a later stage during spring 2022, so the contact parameters were tuned differently. An analogous model according to a Function Analysis for processes could also be built, but since most functions are performed repeatedly, possibly at the same time and not following a given sequence, depending on the movements, activation status, and input from all the users, it can be extremely cumbersome to rearrange the steps according to a process structure. For one such system, the Function Analysis for products is more convenient.

A simplified model of a recurring operation of an IT system is shown in Fig. 7. The user inputs a command onto the keyboard. The keyboard translates the keystrokes into ASCII data that is then transferred to the USB controller. The controller writes data into the memory buffer, and the processor processes the command. The result is transformed by a graphic controller, which generates the pixel map for display. Finally, the information is presented to the user on the screen, closing the interaction loop.

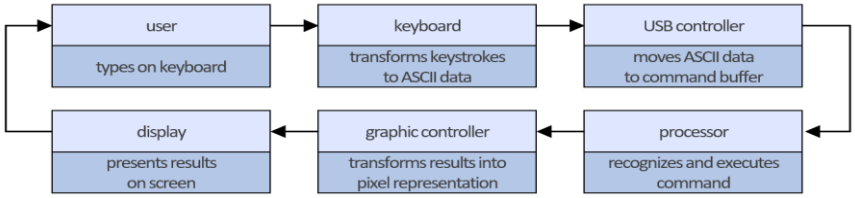

Fig. 7. Simplified information flow model of an IT system operation.

This is a simplified example. One can easily drill it down to further levels of abstraction or focus on specific disadvantages. For instance, a harmful flow may be indicated in the model to reflect possible crosstalks from the supersystem resulting in distorted transmission between the keyboard and USB controller. Parasitic flows may also be added to model possible eavesdropping of the keyboard data, or excessive visibility of the display resulting in the unintended revealing of the screen contents to bystanders, gray zones or bottlenecks may be indicated on the way between USB controller and display to model performance limitations, etc.

7 Summary and further work

We presented a justification for considering software and data as legitimate field-type components of technical systems. This is an important change of approach since it allows to model and analyze IT-related functions of complex systems in a holistic way. It seems essential because many modern systems coming from mechanical or electrical engineering domains (ranging from headphones to cars) are invaded by Information Technology to a level that makes them vulnerable to cyberattacks like other computer systems so that hacking e.g. a Bluetooth-controlled toilet is a viable scenario.

With a whole system modeled in a unified way, one can apply seasoned TRIZ tools, such as Function Analysis and Flow Analysis, to identify functional disadvantages and flow disadvantages, including those regarding software or data. The modeling of the flow and the characterization, as well as type and degree of fulfillment, allows for a further step to specifically solve contradictions/problems in the flow. For this purpose, the tools of flow enhancement are provided by the TRIZ toolbox.

Consequently, an IT system with hardware components and the flow of information can be modeled. Using the Flow Analysis will end up in larger software and data sizes to a very big and, therefore, hard to manage model. For this reason, it is recommended to use Flow Analysis when focusing on a specific aspect and Function Analysis for the broader architectural perspective. It is comparable to Function Analysis for a system and Substance-Field modeling for a specific problem.

References

1. Souchkov, V.: Glossary of TRIZ and TRIZ-related terms, version 1.2, MATRIZ 2018, https://matriz.org/wp-content/uploads/2016/11/TRIZGlossaryVersion1_2.pdf, last accessed 2022/03/19.
2. Selected Topics for Level 1 Training, MATRIZ 2020, https://matriz.org/wp-content/uploads/2020/04/Selected-Topics-for-Level-1-Training.pdf, last accessed 2022/03/19.
3. Ikovenko, S.: Level 3 Training Manual. 2019.
4. Rubin, M., Kuryan, A., Rubina, N., Shchedrin, N., Eccardt, O.: A brief glossary of basic concepts and terms of TRIZ (in Russian), https://triz-summit.ru/onto_triz/100/, last accessed 2022/03/19.
5. VDI 4521 – Part 1. Inventive problem solving with TRIZ - Fundamentals, terms and definitions, Verein Deutscher Ingenieure e.V. 2021.
6. Korolev, V.A.: On dogmatism in TRIZ (in Russian), TRIZ Developers Summit 2017. https://triz-summit.ru/confer/tds-2017/article/, last accessed 2022/03/19.
7. Govindarajan, U.H., Sheu, D.D., Mann, D.: Review of Systematic Software Innovation Using TRIZ. Int. J. Systematic Innovation, 5(3), 72-90 (2019).
8. Beckmann, H: Function Analysis integrating Software Applications, TRIZ Future 2013 Proceedings, pp. 473-482.
9. Souchkov, V.: Extended Function Analysis - Overview. Presentation delivered during German Expert Day 2019. Sulzbach-Rosenberg, German. February 2019.
10. Petrov, V.: Using TRIZ tools in IT. TRIZ Developers Summit 2019. https://triz-summit.ru/file.php/id/f304862-file-original.pdf, last accessed 2022/03/19.

11. Beckmann, H.: Method for Transferring the 40 Inventive Principles to Information Technology and Software, Procedia Engineering 131 (2015) 993 – 1001.
12. Goethals, F.: 20 Trends in Digital Innovations: A TRIZ-trend-structured overview of new business information technologies and innovative applications. Amazon Digital Services LLC, 2014, ASIN: B00P6GNE88.
13. Lady, D.: Information Technology System Cookbook: Introducing TrizIT: TRIZ for Information Technology. AIMS Publications 2013. ISBN 978-1478302513.
14. Mishra, U.: TRIZ Principles for Information Technology. CreateSpace 2010 ISBN 978-818465184.
15. Padabed, I.: Contradictions and Laws of Evolution in Information Systems. TRIZ Developers Summit 2019. https://triz-summit.ru/file.php/id/f304852-file-original.pdf, last accessed 2022/03/19.
16. Altshuller, G.: Creativity as an Exact Science: The Theory of the Solution of Inventive Problems, CRC Press 1984.
17. Souchkov, V.: Development of functionality in technical and business systems (in Russian). TRIZ Developers Summit 2020. https://r1.nubex.ru/s828-c8b/f3244_06/Souchkov-TDS-2020-Functionality.pdf, last accessed 2022/03/19.
18. Lyubomirskiy, A., Litvin, S., Ikovenko, S., Thurnes, C.M., Adunka, R.: Trends of Engineering System Evolution (TESE) – TRIZ paths to innovation, TRIZ Consulting Group 2018.
19. Petrov, V.: Patterns of development of artificial systems (in Russian), TRIZ Developers Summit 2020. https://r1.nubex.ru/s828-c8b/f3215_04/Petrov-TDS-2020-regularities%5b1%5d.pdf, last accessed 2022/03/19.
20. Nähler, H., Gronauer, B., Bertoncelli, T., Beckmann, H., Chrząszcz, J., Mayer, O.: Modeling Software in TRIZ, accepted to TRIZ Future 2022 Conference.
21. Corona-Warn-App. https://apps.apple.com/pl/app/corona-warn-app/id1512595757, last accessed 2022/03/19.

Acknowledgments

The authors gratefully acknowledge Dr Oleg Abramov and Mr. Dmitriy Bakhturin for their explanations regarding the philosophical roots of TRIZ.