Solving data collection tasks in heterogeneous distributed software intensive system using model approach

S. A. Abbas¹, A. I. Vodyaho¹, N.A. Zhukova² and M.A. Chervontsev³

¹Saint Petersburg Electrotechnical University “LETI”, Saint Petersburg, Russia
²St. Petersburg Federal Research Centre of the Russian Academy of Sciences (SPCRAS), Saint Petersburg, Russia
³Research and Engineering Center of Saint Petersburg Electro-Technical University, St. Petersburg

E-mails: saddamabbas077@gmail.com, aivodyaho@mail.ru, nazhukova@mail.ru, chervontsev.mikhail@nicetu.spb.ru

Abstract. The article discusses possible approaches to data collection and information collection in multi-level distributed systems. It is suggested to use model based approach. The requirements to these models are analyzed and the system of models is proposed. The system of models describes the structure and behavior of the observed system in terms of graphs. The resource graph, the data flow graph, the control flow graph, and the query flow graph are used. These graphs are automatically constructed on the basis of information received from the observed system in the form of log files. A virtual machine that implements this approach is proposed.

Introduction
The rapid development of technology, in particular, nanotechnology, telecommunications, software engineering allows creating systems of essentially higher level of complexity in comparison with existing systems for reasonable money and time. One can observe not only increase of the complexity of systems, but also increase of their level of intelligence, which is achieved by extension of the information component. So, it is possible to refer these systems to the class of software intensive systems (SwIS) [1]. Another important trend is the constant expansion of the field of application of SwIS. This expansion is largely due to the low-price segment of SwIS and SwIS which require a high level of reliability. In the first case, the hardest requirements are requirements for minimizing the total cost of ownership, and in the second case, it is the requirements related to reliability. The majority of modern SwIS are multi-level heterogeneous systems consisting of a large number of elements of different physical nature, such as mechanical, electromechanical, natural systems, and people [2]. Such SwIS can realize complex behavior and are capable of learning and self-learning in the future and the structure and behavior of such SwIS may change over time. In modern SwIS a wide variety of tasks are solved, in particular, management tasks, data collection for subsequent analysis, etc. The effective solution of all these tasks requires the implementation of data collection (DC) (monitoring) procedures for both the Observed Systems (ObS) and the Data Collection System (DCS). In this context, the task of DC becomes most urgent and nontrivial, since the observer (O) does not have complete information about the structure of the ObS and its business processes (BP).
1. The main types of DC tasks, the solution of which is associated with the construction of models of the ObS

Solving many classes of problems related to DC requires building an ObS model. Models are usually built to solve the following 5 classes of tasks.

Class A. Processing of observation results. The realization of transformations of the raw data from observed objects and forming needed views.

Class B. Restoring object models from existing data. The definition of the structures of objects and their changes according to the results of monitoring.

Class C. Monitoring management. Determining the types of data to be collected and defining the monitoring procedures parameters.

Class D. Building views of object models. Object models are presented in the form required for solving practical tasks. Views can be formed to identify deviations in the state of objects, their localization, and determine the causes of special situations occurrence.

Class E. The study of the statistical monitoring data. Building statistical models of objects and environments for a given subject area.

Tasks of classes A and B are related to data processing and analysis, tasks of class C are related to monitoring, and tasks of classes B and D are related to building models of complex objects.

2. A generalized DC problem statement

In the most general form, the DC problem can be defined as the process of obtaining the required data, information or knowledge (DIK) taking into account available knowledge about the ObS and the Data Collection Procedure Results (DCPR): DIKti = f(DIKti-1, DCPR). In turn, the DCPR is formed on the base of the goals of the DC and the available knowledge about the ObS: DCPR = f (NDIK, DIKti ), where NDIK is needed DIK. NDIK can be obtained either as a result of DCPR operation, or as a result of transformation (transformation or transformation chains T) of the results of DCPR operation.

\[
\text{DCR} \xrightarrow{T} \text{DCPR} \xrightarrow{f} \text{DCR}.
\]

where DCR are data collection results.

The DC problem can be formulated in terms of 8 entity classes (Figure 1) DCS = <SH, Obs, DCSS, DSLRQ, DSLRS, POL, SCR, LOGS>, where DCS is a data collection system, SH is a set of observers (stakeholders, SH), ObS is an observed system, DCSS is a DC Subsystem, DSLRQ, DSLRS is a set of requests and a set of responses formulated in terms of a domain specific language (DSL) [3], POL is a set of policies, SCR is a set of scripts, LOGS is a set of log files or just logs. (For the sake of simplicity, we will use the term DCS instead of DCSS when this is not ambiguous.)

![Fig 1. Generalized structure of the DCS.](image)

The DC problem can be divided into two main subproblems: the formation and execution of the requests and the presentation of the responses in terms of DSL. The first task involves performing transformations DSLRQ → Vu, the second subtask assumes transformation Vk → DSLRS, where Vu is a vector of parameters to be determined as a result of realization of DC procedure, and Vk is a vector of values of these parameters. The Vk vector can be considered as a Vu vector populated with parameter values.

DSLQ and DSLRS are the DSL requests and DSL responses. (Note that different user groups may use different DSLs.) The second subproblem involves the transformationVu → Vk, Vk = f(Vu,
ObSP), where ObSP are the parameters of the ObS and some of the parameters of ObSP may be unknown.

3. DC procedure

The DC procedure can be described as a business process (BP) or as an automaton.

The DC procedure can be defined as a transition from the vector X to the vector Y: \( X \rightarrow Y \), where X is the input vector and Y is the output vector. The input vector X contains an arbitrary finite number of elements, each element can be either simple or complex (includes an arbitrary number of simple or complex elements). The number of nesting levels can be any, but not infinite. A vector of parameters of any length is assigned to each element \( v_i \), which has at least 3 attributes: identifier ID, parameter value and reference to the source, where ID is the unique identifier of the element, parameter is the value, and link is a way to get information about the value of the parameter. The parameter value is a DIK of arbitrary complexity. The parameter value can have a specific value \( v_i = \text{known} \) or an undefined value \( v_i = \text{unknown} \). A set of elements of the \text{known} type can be considered as a priori knowledge about the ObS. All elements of the output vector Y have the values \( w_i = \text{known} \). The reference field specifies how to get information about the element. This can be \(<\text{Reference}> : = <\text{Data source}> | <\text{Rule Set}> | <\text{Procedure}>\).

In the special case, each element X corresponds to an element Y that has similar attributes. This is true when the ObS structure is static. If the ObS structure is dynamic (i.e., new elements may appear and old elements may disappear during operation), then the structure of the X and Y vectors does not match. In relation to the subject of our research, the second case is the point of interest.

The values of the parameters of the output vector Y are used to form the required representations. In order to transform initial vector X to destination vector Y very often it is necessary to do several steps. In this case we have a chain of transformations of the form \( X \rightarrow T_1 \rightarrow T_2 \rightarrow T_i \rightarrow T_k \rightarrow Y \rightarrow P \), where \( T_i \) – the results of intermediate transformations, and P – the required representation. This chain defines the order in which the required values are defined.

The automata representation of the DC procedure is shown in Figure 2. Logs are sent to the automaton's input. In this case, the automaton switches from the current state \( S_t \) to the next state \( S_{t+1} \). The output P is a certain presentation (view) that is formed in accordance with the user's request: \( P_{t+1} = f(St, \text{log}, P) \), \( P=f(St, RQ) \), where \( St \) is a current state, \( P \) is a needed presentation, \( RQ \) is a user request.

![Fig 2. Automata representation of the DC procedure.](image)

There are 3 basic variants of DC procedure organization (DC strategies): direct DC, model-based DC, and mixed DC strategies.

The direct DC procedure can be defined as \( V_k = f(V_u, \text{Obsp}) \), where \( V_k \) is the vector of parameter values, \( V_u \) is the vector of parameters to be determined, and ObSP is the known parameters of the observed system. A generalized sequence of actions for implementing the direct DC procedure is shown in Figure 3. When a request is received, a script is generated based on the knowledge about the ObS and the DSLRS is generated based on the results of its execution. When a new request appears, the required information about the ObS is searched again.

![Fig 3. Direct strategy.](image)
The model strategy, unlike the previous one, is characterized by the fact that when a DSLRQ is received, it is not addressed to the ObS, but to its model. However, there is a separate process that is responsible for keeping the model up to date. The structure of the generalized procedure that implements the model approach is shown in Figure 4. The procedure includes 2 parallel and asynchronous processes: the process of executing user requests and the process of keeping the model up-to-date.

These processes can be represented as follows: $V_k = f_1 (V_u, ObSM)$, $ObSM = f_2 (ObSP)$, where $ObSM$ is the ObS model.

According to the generalized DC procedure that is shown in Figure 4, the monitoring process is continuously running in the system. This process is responsible for keeping the ObS model up-to-date. When a request DSLRQ from the user is received, the ObS model is accessed. Since the model changes all the time, and different groups of users use their own DSL, there is a need for some intermediate query representation language. Policy mechanisms can be used as this language, which will be discussed below.

The use of the model approach can reduce delays when executing the query. This solution can be considered as a kind of two-stage pipeline. Compared to direct DC, the model approach has two main advantages: the ability to reduce response time and the ability to work with historical data. The main drawback is related to the need to build and store models. As the size of the systems increases, it becomes impossible to have a complete model and keep it up-to-date.

Mixed strategy is a combination of the strategies discussed above. The idea is that the ObS upper level uses a model strategy, while the ObS lower level implements direct DC. The usage of this approach allows, first of all, reducing the size of the model to an acceptable size. This approach can be described as follows. The vector of parameters to be determined is divided into 2 subvectors: the
vector of parameters that can be obtained directly from the ObS model and the vector of parameters that can be obtained with the help of ObS resources polling: \( V_k = f_0(V_{hu}, V_{lu}) \), \( V_{hk} = f_1(ObSM) \), \( V_{kl} = f_2(ObSP) \), where \( V_k \) is the vector of parameters to be determined, \( V_{hu} \) is the vector that can be obtained by means of the ObS model querying and \( V_{lu} \) is the vector of parameters that can be defined using polling procedures, \( ObSM \) is the ObS model and \( ObSP \) is the ObS polling procedure.

4. ObS Models

The DCS model can be represented in 2 ways. In the first case, DCS and ObS are separate systems. ObS and DCS are designed independently of each other. ObS and DCS have their own users. Observers are not the users of DCS and vice versa. In this case DCS and ObS can be considered as a loosely coupled system. Usually, the bandwidth of the channels connecting the ObS and DCS have limited bandwidth, in particular, there may be the only one communication channel. This variant can be defined as stand alone DCS.

In the second case, DCS and ObS can be considered as a single system. In other words ObS and DCS form strongly coupled system. In this case DCS observers are ObS user and vice versa. The ObS and DCS are designed simultaneously, and the interests of observers are taken into account when selecting the DCS architectural and structural solutions. ObS and DCS can be linked by many communication channels. A separate subsystem of the DCS can be integrated in the ObS subsystem and both subsystems can use the same resources. This option can be defined as built in DCS.

As a typical example of a modern intelligent heterogeneous ObS, one can consider cyber-physical systems (CPS) that implement the Ambient intelligence (AmI) paradigm [4] built on fog computing platforms [5]. Many modern real systems can be considered as profiles of such a system. The generalized structure of AmI CPS and its interaction with DCS is shown in Figure 6.

![Fig 6. Generalized AmI CPS structure.](image)

AmI CPS can be defined as multilayer system which can be described at 6 levels (Layers): sensor level, the level of fog computing, the level of cloud computing, the level of individual CPS, the level of systems of CPS (SoCPS) and the ambient intelligence level.

The sensor level, fog&mist level and cloud level correspond to the levels of the fog computing reference model [5]. Many modern CPS are built on the fog platform, and for the CPS the fog platform is represented as a set of services. The system can consist of any number of CPS that can be integrated at different levels (data, applications, user interfaces). At each level, models are described using their own dictionary.
5. Monitoring objects

In the Figure 7 main monitoring objects linked to the levels of the AmI CPS model are shown. These can be physical devices, physical objects, virtual devices, virtual objects, and business processes. These entities of observation can be directly or indirectly observed. In the first case, the parameter value is read from a certain port or access point. In the second case, the required DIK are determined with the help of a rule set or a procedure.

![Diagram of monitoring objects](image_url)

6. Generalized DCS structure

The generalized DCS structure is shown in Figure 8.
The DCS consists of 4 asynchronously functioning virtual machines (processors): a log processor, a model processor, a script and policy processor, a DSL processor and a DIK repository. All the listed elements are distributed. The log processor is responsible for picking up logs by implementing scripts and pre-processing logs. The model processor implements two main functions: maintaining the adequacy of models and generating responses to user requests. The script and policy processor is responsible for creating scripts based on policies. DSL processor is an interpreter for a number of DSLs that different categories of users work with.

At the sensor layer, the DC procedure is realized by means of script execution and pre-processing of the information contained in logs.

The edge layer is divided into two sublayers: fog level and mist layer. At the mist layer, procedures related to the real time are realized. Usually at this layer events in the ObS that require an immediate reaction are processed. At this layer, virtualization is not used, and the controllers built into the ObS subsystems are usually used as resources. At the fog layer, there are usually quite powerful local servers that can be used for running virtual machines and local storages. At the fog layer the following elements are deployed: a local (low level) model processor, a local DIK repository and a local script and policy processors.

Cloud level can be implemented on different types of clouds (public, private, hybrid). This layer hosts global (high level models), model processor, script and policy processor, and all services that do not require real time operation and wide band communication channels.

The CPS layer, SoCPS layer and AmIS layer form the application layer. CPS is a collection of subsystems of different nature. These can be physical systems, software systems that implement BPs, or virtual machines.

The SoCPS layer is first of all an integration layer where individual CPS are integrated into a system. It can be done at data, BP or user interface levels.

The user interface integration for the most part is implemented at the AmI layer.

DCS according to Figure 8 is focused on the implementation of a mixed strategy given in Figure 5. DCS implements 2 main subprocesses: the subprocess of keeping the model up-to-date and the subprocess of forming responses to user's requests. The generalized algorithm for these subprocesses is the following.

Algorithm for keeping the model up-to-date.
1. Starting the ObS monitoring procedure
2. Receiving and preprocessing logs
3. If it is necessary then correct model else go to item 2

User requests processing algorithm
1. Waiting for a request from a user
2. Request acceptance and realization of DSL ➔ Model Query Language (MQL) transformation
3. Request to the ObS model using MQL.
4. If the response is not received, then continue, else realize MQL ➔ DSL transformation and go to item 1
5. Policy definition
6. Generate script from the policy

![Fig 8. Generalized DCS structure.](image-url)
7. Execute script
8. Add necessary fragments to the ObS model
9. Query the new ObS model.
10. Realize MQL \rightarrow DSL transformation

7. ObS models representation
This is one of the key moments, since ObS models are subject to strict requirements, the main of which are the following: the model must adequately describe the current, past and future states of the OS in terms of its structure and behavior, it must be possible to automatically build and correct the model on the basis of information about events in the ObS received in the form of logs, the model must provide a shorter response time to user requests compared to direct DC.

The most stringent should be the requirement for automatic construction of structural and behavioral models. The following well-known approaches were considered as candidates for using the ObS model: the actor approach [6], G-graphs [7], and a model approach based on management strategies [8]. The last model was chosen. In relation to DCS this model can be interpreted as follows: OSM = \langle RG, RGA, DFG, CFG, QFG, BPA \rangle, where OSM is the ObS model, RG is a resource graph, RGA is a resource graph control automaton, DFG is a data flow graph, CFG is a control flow graph, QFG is a query flow graph, BPA is a restructuring automaton. The RG describes the current structure, and the RGA defines the possibilities for restructuring. BP are described in terms of the DFG that describes data dependencies, the CFG describes the order in which operators are executed, and the QFG describes the problem to be solved in terms of requests for operator execution.

In [9, 10] the questions of automatic construction (synthesis) of multi-level structural models of ObS are considered. Within the framework of the process mining concept [11], the problem of automatic construction of the CFG and partially the DFG is solved. It should be noted that the presented ObS model is very simplified, since it does not take into account the fact that the ObS is a multilevel system. This model can be implemented on different platforms, in particular, as a compiled executable file, or on an ontological platform, or using knowledge graphs. The analysis shows that the highest performance can be obtained when implementing the model using high-level languages, and the simplest solutions from the point of view of programming can be obtained when an ontology approach is used.

8. Problems in implementation of the suggested model approach
At the conceptual level, there are three alternative approaches to building a DCS. In the first case, the DCS is constructed as an element of the control system. In this case as a rule the number of stakeholders is minimal, the main one is the subsystem that is responsible for reaction generation, and the main requirements are associated with the minimum response time. In this case, the DCS itself can be embedded, for example, in the equipment. In the second case, the DCS is built as an information management system, the main purpose of which is to collect data and present it in different formats to different groups of observers in accordance with their roles. In this case, the main load usually falls on the query processing subsystem and the subsystem of views formation. In the third case, the DCS is constructed as a context-aware system [12], which has a separate model describing the current context. This solution is usually used in cases where it is necessary to analyze and take into account a constantly changing situation and react quickly to it, and the observer wants to get information on the situations.

From the point of view of architecture service-oriented architectures are preferable, first of all, microservices. If DCS are not focused on working in real time, then there are no essential problems. However, if the ObS is running even in soft real-time mode, then one can meet problems with performance. The bottlenecks are the models and communication channels through which client-service interactions are implemented when using even microservices. In principle, one can increase the speed of the model processor without using the ontologies, but this leads to more complex programming and, consequently, to an increase in the cost of development.
In the most general form, the service representation of DCS is defined as $SRVP = <\{srv\}, \{script\}>$, where $\{srv\}$ is a set of services, and $\{script\}$ is a set of scripts that perform operations on services. There are two main groups of services: infrastructure services (low level services) and high level services. Scripts are programs in DSL or a general purpose language.

9. Case study

The proposed model was used by the authors to solve a number of practical tasks in different subject domains. One of such examples is CTV monitoring system.

Modern CTV is a distributed large scale system with hundreds of thousands and even millions of subscribers. CTV includes data transmission networks, server equipment, and client equipment and subscriber equipment. Recently, CTV owners began to face serious problems associated with a decrease in the number of subscribers due to competition from, in particular, Internet television.

In the event of a malfunction on the subscriber's side, he can inform the operator about the problem in different ways (via the Internet, via the phone). The operator has a system of the Service desk type, where the request is entered. Support specialists identify the place, time and causes of malfunctions, and restore the operation of devices. The work can be performed remotely or directly by users. This procedure usually takes from several hours to a day.

Until a certain time, this method of maintaining networks turned out to be acceptable. The main business requirements were the following: i) a radical increase in the availability indicator, which should be achieved by reducing the average system recovery time from several hours to several minutes; ii) reducing the total cost of ownership, primarily by reducing the number of service personnel; iii) smooth transition to a new system: no need to stop the whole CTV system to switch to a new system, the ability to use the user's existing CTV, the ability to use existing communication channels.

As a solution to the listed problems, it was proposed to use a model approach. Global models were hosted on a central server, local models were hosted on local servers. As a result, it became possible to detect the causes of user problems in an automatic mode in 98% of cases, which in turn reduced the recovery time from several hours to several minutes.

The detailed description can be found in [13].

Conclusion

The proposed approach is quite generalized and in most cases it is not necessary to use it in full scale. The main field of application should be considered as complex distributed CPS with a constantly changing structure and dynamic business processes. The undoubted advantage of the proposed approach is the ability to execute DC procedures in the frames of CPS with high level of complexity and intelligence. The authors associate the further development of the proposed technology with the use of cognitive models.

References

[1] Lattanze, Anthony J., Architecting Software Intensive Systems. Practitioner’s Guide. Taylor & Francis Group, LLC Boca Raton London New York FL, 453 p., 2009.
[2] Sanfelice, R.G., Analysis and Design of Cyber-Physical Systems. A Hybrid Control Systems Approach // Cyber-Physical Systems: From Theory to Practice / D. Rawat, J. Rodrigues, I. Stoimenovic, CRC Press, 2016, ISBN 978-1-4822-6333-6.
[3] Fowler, M., Domain-Specific Languages, Addison-Wesley. Upper-Saddle River. NJ, 2014, 583 p.
[4] Korzun, D., Balandina, E., Kashevnik, A., Balandin, S., Viola, F., Ambient Intelligence Services in IoT Environments: Emerging Research and Opportunities; IGI-Glbal, 2019, 199 p.; DOI: 10.4018/978-1-5225-8973-0.
[5] Open Fog Reference Architecture for Fog Computing. Available online: https://iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf (accessed on 07.09.2020).
[6] Agha, G.A., Actors: A Model of Concurrent Computation in Distributed Systems: MIT. MA, 1986, 142 p.
[7] Okhtilev M.Yu., Sokolov B.V., Yusupov R.M., Intelligent technologies for monitoring and controlling the structural dynamics of complex technical objects, Moscow: Nauka, 2006, 410 p.

[8] Vodyakho A.I., Smolov V.B., Plyusnin V.U., Puzankov D.V., Functional-oriented processors, Smolova. V.B., Ed., Leningrad: Mashinostroenie, Leningrad. dep., 1988, 224 p.

[9] Osipov, V. Yu., Vodyaho, A. I., Zhukova, N. A., Tianxing, M., and Lebedev, S., Distributed Technical Object Model Synthesis Based on Monitoring Data, International Journal of Knowledge and Systems Science (IJKSS), 2019, Vol. 3, pp. 27–43.

[10] Vodyaho, A. I.; Osipov, V. U.; Zhukova, N. A.; Chervontsev, M. A., Cognitive Technologies in Monitoring Management, Automatic Documentation and Mathematical Linguistics, 2019, Vol. 53, Iss. 2, pp.71–80; DOI:10.3103/S0005105519020080.

[11] Van der Aalst, W., Process Mining Data Science in Action, Second Edition, Heidelberg. Springer, 2016, 468 p.

[12] Loke, S., Context-Aware Pervasive Systems. Architectures for a New Breed of Applications, Taylor & Francis Group, Boca Raton, FL, 2007, 220 p.

[13] Osipov, V; Zhukova, N; Vodyaho, A. About one approach to multilevel behavioral program synthesis for television devices, Int. J. of computers and communications, 2017, Vol. 11, pp.17–25.