Igor KOROPIECKI[1], Krzysztof PIOTROWSKI[1], Robert SZULIM[2]
[1]IHP – Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany
[2]Uniwersytet Zielonogórski Instytut Metrologii, Elektroniki i Informatyki

SMARTDSM: DATA SPACE MIDDLEWARE FOR DISTRIBUTED MEASUREMENT SYSTEMS

The article describes a platform that consists of multiple modules which together allow to accelerate the process of setting up a secure communication and data storage solution. The implementation approach makes it possible to apply the platform in many deployments that can be interconnected on the data and communication layers.

SMARTDSM: PLATFORMA MIDDLEWARE DO AGREGACJI DANYCH DLA ROZPROSZONYCH SYSTEMÓW POMIAROWYCH

Artykuł przedstawia platformę, która może zostać użyta do przyśpieszenia procesu tworzenia systemu wymagającego bezpiecznej komunikacji oraz przechowywania danych. Implementacja platformy pozwala na zastosowanie w wielu systemach, które mogą być ze sobą połączone na poziomie warstwy danych oraz komunikacji.

1. INTRODUCTION

In order to transfer and store data, measurement systems require some sort of communication and data storage solution. In most cases, security and access control are also mandatory. For a simple centralized measurement system in which data ends up on a single device, there are plenty of solutions available (see Chapter 2). The task gets hard when the system has to be decentralized, since not all solutions are designed to support it. The task gets even harder when interaction with other existing and future systems is required or the entities are not simultaneously present in the network.

Every system deployment comes with a set of factors that have to be considered when choosing a platform. When focusing on the environment, some systems might operate on powerful hardware while other systems might require support for operation on hardware with low computational capabilities. Also, the network access has to be taken into account, systems might operate in private networks that use NAT-enabled routers which prevent incoming connections. When focusing on the system characteristics, some systems might require simple reading and writing, while other systems might need a near real-time communication solution to notify the network about occurring events.

Many deployments end up using existing platforms that match most of the requirements and implement the missing parts. This works only for systems that work in a pattern that the existing solutions were designed to address. Choosing the right solution impacts the success of the system deployment and any related future projects which might end up being locked to a particular platform.

An ideal platform should offer multiple blocks to choose from, in order to ease building the system as much as possible and minimize the use of external solutions. Each external solution requires additional effort to include and configure, it also introduces potential compatibility errors that could be expensive to fix, for example in remote or spread-out deployments. The platform should be generic and easily understandable but also not too plain and crude which would result in more code written than it is necessary. Ideally, the mechanisms should derive from concepts that are already known. The security should be strong and easily configurable, the communication channel should be immune to MITM (man-in-the-middle) attacks and also guarantee the identification (and verification) of entities participating in communication. The platform should offer access control and data ownership features. Future-proofing should be kept in mind to quickly replace security algorithms that are no longer safe.

2. RELATED WORK

The topic of providing communication and data storage is the backbone of computer science. Almost any system ends up sending or storing data. Because of that, there are many platforms and frameworks that can be used to create a solution.

The base for most tuple space middleware platforms is the Linda coordination model [1] which defines a simple set of operations available in the tuple space – rd, in, out and eval. The clients communicate through writing and reading tuples (finite ordered lists of elements) to a shared memory space (the tuple space). Most notable platforms and frameworks that implement this model are: JavaSpaces, Tupleware, Klava, CryptoKlava, MozartSpaces and GigaSpaces, from which only the MozartSpaces and GigaSpaces (commercial) are still maintained. Other notable platforms are: WuKong [2] which offers a flow-based graphical programming environment for IoT deployments, Civitas [3], which offers a low footprint service-oriented middleware, OpenIoT [4] which offers a local and cloud-based generic middleware platform and SmartCityWare [5] which offers a cloud-based and service-oriented platform with various functionalities such as discovery and broker.

3. PROPOSED APPROACH

The proposed platform offers a set of modules that can be configured and connected to establish a lightweight platform for secure data processing. The platform serves as a proving ground for further enhancements related to protocols, security, reliability and data integrity. To be able to move in any direction with the development, the authors believe that a well-known and extensible base that can be further improved is necessary. The platform has been created in Java and web technologies in order to support various operating systems and hardware. The communication security is based on the public key infrastructure (PKI) [6] that has been implemented by using Java's Secured-Socket Extension (JSSE) [7]. Each communicating entity (service, server) has a certificate that is used for encryption and identification. On top of the sockets, there are several layers (Fig. 1) that provide communication. The Byte layer instantiates and manages the connection lifecycle of supported transports (real-time, polling). The Proto layer provides transparent access to the Byte layer. The upper smartDSM layer represents the entities (service, server, proxy, discovery, etc.) that use or provide functionalities.

The systems based on the smartDSM platform can be created by implementing services. A service in the context of the platform is a standalone Java application that uses the platform-provided API to implement system-specific functionalities. The Fig. 2 shows a single service that is always connected to a middleware server (referred to as local server) and relies on this server when communicating and storing data. The local middleware server is connected to other providers available in the network (proxy, discovery, cert. authority - CA) and other middleware servers (referred to as remote servers).

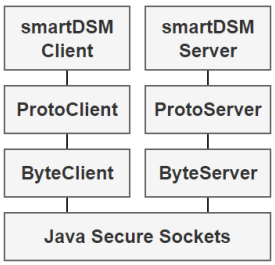

Fig. 1. Communication stack
Rys. 1. Stos komunikacji

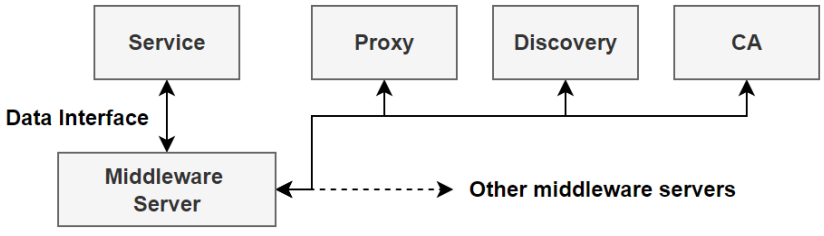

Fig. 2. Example system architecture based on smartDSM
Rys. 2. Przykładowa architectura systemu opartego o smartDSM

When working with data, smartDSM operates on variables and stakeholders to store data and identify ownership. A variable in the context of the platform is a structure (tuple) with metadata (name, limits, sub-fields and types) that can contain any amount (limited by disk space) of stakeholder data (values,

ownership, timestamp, source). A stakeholder in the context of the platform is an entity (person, company) that owns an asset in a system built on top of smartDSM. Any service runs on behalf of a specific stakeholder and any performed actions are executed in the context of this stakeholder. The Fig. 3 presents two services belonging to different stakeholders that write data into a variable. The middleware server processes the requests and writes data into the variable, annotating which values belong to which stakeholders. At any point in time, stakeholders can be granted permission to access data of other stakeholders. The writing stakeholder can specify the types of access to grant – read, write, update or clear. Later, the permission can be revoked.

Each method that outputs a list of objects (variables, data, permissions, subscriptions), supports pagination (page number, items per page and sorting options). Additionally, when reading variable data, it is possible to specify a filtering query that is compatible with the pagination. For notifications about variable modifications, the platform offers a subscription mechanism. The notification is sent when the data changes or periodically, at a configurable time rate. Available modules in the platform:

- **Server** – a standalone Java application that processes requests sent from services. Implements the access control, data storage and communication with other middleware servers and modules. It can work as a single communication and storage hub or as a node in a hierarchical grid.
- **Data Interface** – a Java library that can be embedded in a service that is intended to communicate with a middleware server. Defines available actions that can be grouped as following: a) variables – list, info, read, write, update, clear, b) permissions – list, grant, revoke, c) subscriptions – list, create, remove, d) general – system status. Also, for increased performance, bulk variable reads and writes are supported.
- **Proxy** – a standalone Java application that enables communication between middleware servers that are not directly visible in the Internet.
- **Discovery** – a standalone Java application that allows the middleware servers to announce their presence and information about the data stored on a particular server.
- **Web Certificate Manager** – a website that allows to coordinate security credentials for specific projects that run on the smartDSM platform. The manager communicates with a separate server that is only used to issue certificates.
- **AdminGUI** – allows to manage the middleware servers. Offers a set of widgets that can be used to manage data, view clients and diagnostic information, configure permissions and run benchmarks.
- **MWGUI** – an example service based on the Data Interface, intended for non-technical end users of the platform. Allows to browse data and modify permissions from the perspective of the end user.

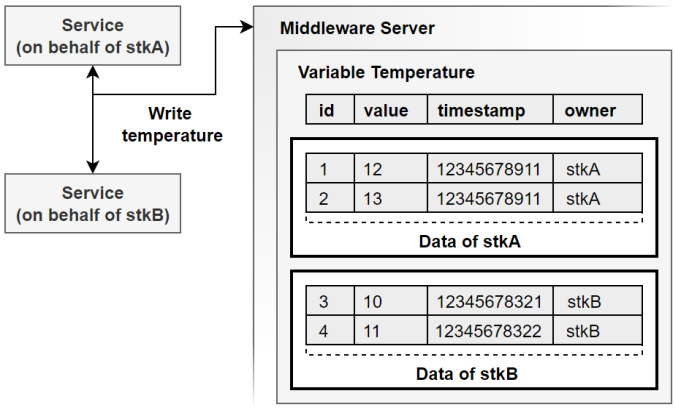

Fig. 3. Services writing data into the middleware
Rys. 3. Serwisy zapisujące dane w middleware

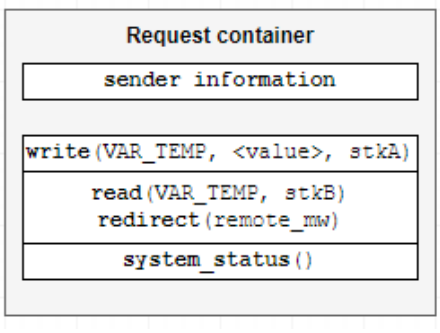

Fig. 4. Request container with partial requests
Rys. 4. Kontener zawierający zapytania częściowe

The Data Interface allows to send multiple requests at once. The Fig. 4 presents a request container that includes the sender information (service, stakeholder) and the partial requests (write, read,

system_status). In the example, the read request contains an optional redirect directive which tells the local middleware server to pass this particular partial request to a different middleware server.

4. CLOSING REMARKS

The proposed middleware platform allows to deploy a system that requires secure communication, data storage and exchange. The system can use discovery, proxy and authorization mechanisms to find data of interest, keep working in NAT-enabled networks and authorize and verify the identity of peers that it interacts with.

The platform described in this article is a work in progress. The future plan is to finish each provided functionality and to develop mechanisms related to data integrity, reliability and high availability. The work will result in a PhD thesis of the first author, on *High availability middleware platform for embedded systems*.

ACKNOWLEDGMENTS

This work was supported by the European Regional Development Fund within the BB-PL INTERREG V A 2014-2020 Programme, "reducing barriers - using the common strengths", project SmartRiver, grant number 85029892 and by the European Union ebalance-plus project under the H2020 grant number 864283. The funding institutions had no role in the design of the study, the collection, analyses, or interpretation of data, in the writing of the manuscript, or in the decision to publish the results.

REFERENCES

1. Tolksdorf R., Bontas E. P., Nixon L. J. B.: Towards a tuplespace-based middleware for the semantic Web. The 2005 IEEE/WIC/ACM International Conference on Web Intelligence, 2005.
2. Shih C., Chou J., Lin K.: WuKong: Secure Run-Time environment and data-driven IoT applications for Smart Cities and Smart Buildings. J. Internet Serv. Inf. Secur. 2018.
3. Villanueva F. J., Santofimia M. J., Villa D., Barba J. and López J. C.: Civitas: The Smart City Middleware, from Sensors to Big Data. International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS). 2013.
4. LX: OpenIoT – Open-source middleware for the Internet of Things. 2015. Available at: https://lx-group.com.au/openiot-open-source-middleware-internet-things (last accessed on 31.03.2022)
5. Mohamed N., Al-Jaroodi J., Jawhar I., Lazarova-Molnar S. and Mahmoud S.: SmartCityWare: A Service-Oriented Middleware for Cloud and Fog Enabled Smart City Services. IEEE Access - volume 5, 2017.
6. IETF: RFC-3647 – Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework, November 2003.
7. Oracle Corporation: Java Secure Socket Extension Reference Guide. Available at: https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html (last accessed on 15.03.2022)