2025 IEEE/SICE International Symposium on System Integration (SII)
January 21-24, 2025, Munich, Germany

Software-Defined Robotics: Architecture, Information Model, and Implementation *

Hong Seong Park, *Member, IEEE*

Abstract— **Recent advancements in computing, network speeds, and cloud technology have led to the emergence of Software-Defined x (SDx), a concept that promotes a shift from traditional hardware-dependent systems to more flexible, software-defined environments. In the context of robotics, this approach is termed Software-Defined Robotics (SDR), which aims to abstract hardware functions into software, enabling dynamic development and deployment of robotic applications without the need for extensive hardware modifications. This paper proposes the architecture of SDR to enhance flexibility, interoperability, and portability in robotic systems, which is designed based on the information models defined in ISO/DIS 22166-202 and ISO 22166-201. This paper demonstrates the feasibility of SDR using the TurtleBot3 as a case study. The proposed approach is validated by showing two similar but distinct SDR implementations for the TurtleBot3 through Gazebo simulation. The results emphasize the potential of SDR to revolutionize the development and management of robotic systems by reducing hardware dependency and fostering innovation. This paper provides a comprehensive overview of the architecture, advantages, and constraints of SDR and presents an implementation model based on the TurtleBot3, providing insights into future developments in software-defined robotics.**

I. Introduction

Recent technological advancements, particularly in computing, network speeds, and cloud technology, have increased software's relevance and impact across various sectors. This shift is encapsulated in the concept of Software-Defined x (SDx), which signifies a move away from traditional hardware-dependent systems toward more flexible, software-defined environments [1]. SDx separates hardware from software, allowing applications to be developed and deployed directly in the cloud or on embedded controllers without being limited by physical hardware constraints. This concept has been applied in several fields, such as radio [2], networks [3,4], vehicles [5-8], manufacturing [9,10], and robotics [11].

In robotics, the potential application of SDx, referred to as Software-Defined Robotics (SDR), could revolutionize the field by abstracting some hardware functions into software. According to ISO 8373:2021 [12], robots are defined as programmed actuating mechanisms with autonomy for performing movement, manipulation, or positioning tasks. Robots come in various forms, including industrial robots, mobile robots, humanoid robots, and medical robots, each differing in structure, sensor types, and motor configurations. The implementation of SDR would allow robotic applications to be developed and updated dynamically, circumventing the traditional limitations imposed by physical hardware. This flexibility would enable faster development and adaptation of robotic functionalities without the need for extensive hardware modifications.

Robotic systems currently use various middleware, such as the Robot Operating System (ROS) [13], OpenRTM [14], OPRoS [15], and OROCOS [16], which facilitate the management of software modules. Robots also operate on diverse operating systems, including Windows, Linux, and real-time operating systems, suggesting that the software constraints in robotics may be more pronounced than in other SDx applications. Middleware is crucial for managing the increasing number of software modules on a robot, enabling efficient integration and communication between different software components and the hardware they control.

One of the critical aspects of SDx, including SDR, is modular software design, where different software modules need to understand each other's interfaces and the hardware interfaces they access. As the number of software modules grows, there is a need for standardized interfaces to ensure these modules are flexible, interoperable, and portable across different systems. However, standardizing interfaces poses challenges, as it could hinder the evolution of related software or devices. To address this, the ISO TC299 is developing an information model for robot modules, which standardizes the format for providing information about a robot's hardware and software components. This model, detailed in DIS 22166-202 [17] and ISO 22166-201 [18], aims to make developing software for various robotic systems more accessible, promoting innovation and adaptability in the field.

The introduction of SDR emphasizes flexibility, interoperability, and portability by enabling software modules to interact independently of the underlying hardware. This modular approach allows for rapid development and deployment of new robotic functionalities, reducing dependence on specific hardware configurations. The information model standards support this by ensuring that the software can adapt to different hardware setups, facilitating easier updates and maintenance of robotic systems.

Despite these benefits, transitioning to software-defined systems in robotics presents unique challenges. Traditional robotic systems often rely on rigid hardware configurations, which can limit their flexibility and scalability. Existing

*Resrach supported by Robot industry technology development program (No RS-2024-00507746 Development of Framework and Common Interface and Service Operations System for Cloud-Based SDR Common Services.) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea)..

Hong Seong Park is with Dept. of Electrical and Electronic Engr., Kangwon National University, GangwonDo, 24341 Rep. of KOREA (e-mail: hspark@kangwon.ac.kr)

software frameworks and middleware address only a small portion of the market needs, highlighting the need for a more comprehensive approach that supports flexibility, portability, and interoperability.

This paper summarizes advantages and constraints of SDR, reorganized based on [21], when SDx principles are applied to robotics, specifically SDR, to enhance flexibility, interoperability, and portability in robotic systems. It focuses on improving the design, development, and deployment of robotic functionalities by abstracting hardware functions into software. This approach reduces hardware dependency and facilitates the rapid development and adaptation of new features without requiring physical hardware modifications.

This paper demonstrates the validity of SDR using a simple example with TurtleBot [22], showing that it can be implemented through an information model. To do this, the paper presents the actual hardware-related properties of the TurtleBot and illustrates how these properties are mapped to the information model. The proposed approach is validated by demonstrating that two similar but distinct SDR implementations for the TurtleBot3 are successfully operated within a Gazebo simulation [23].

Section 2 presents the hardware-related information for robots that can be utilized for SDR. Section 3 introduces the information models for software modules, while Section 4 proposes the architecture of SDR and discusses its advantages and constraints. Section 5 provides the implementation model for SDR based on the TurtleBot and the results of this implementation. The final section provides the conclusions.

II. HARDWARE-RELATED INFORMATION FOR ROBOT

Robot hardware information includes details such as the mechanism including the motion structure and the actuating modules, the features and poses of the sensors, robot weight, rated speed, and maximum load. The control software modules can vary depending on the type of mechanism. The pose of the robot and obstacles can differ depending on the poses of the sensors mounted on the robot. Additionally, to check the rated speed and load, it is necessary to know the type of motors used in the robot, the robot's weight, and etc. In order to avoid obstacles, the size of the robot has to be known. In other words, various information is needed depending on the purpose of SDR. From an application perspective, all physical information about the actual robot must be included. However, in order to develop the control software modules, only information on the relevant modules such as mechanism and actuating modules may be required. According to SDR's usage these factors must be considered. An example of such information is shown in Table 1, which is required for TurtleBot3 Waffle Pi.

TABLE I. INFORMATION FOR HARDWARE ROBOT

Item	Detailed Item	TurtleBot3 Pi
System Aspects	Size	281(L) x 306(W) x 141(H) (mm)
	Weight	1.8kg
	Max. translational vel.	0.26m/s
	Max. rotational velocity	1.82 rad/s
	Maximum payload	30kg
	Threshold of climbing	10 mm or lower
Mobility aspects	Mechanism(Differential/Steer./ Omnidirectional/Universal/Ackermann Steer./Crab Steering/other)	Differential
	Wheel – radius	33mm
	Wheel – separation	287mm
	Motor – encoder	Absolute. 12 bit
	Motor – Pose(left, right)	(0,0.144,0.023,1.57,0,0) (0,0.144,0.023,1.57,0,0)
	IMU – property	Gyro/Acce/Mag. 3 Axis
	IMU – Pose	(0,0,0.068,0,0,0)
	LiDAR – Distance Range	120 ~ 3,500 (mm)
	LiDAR – Angular Range	360°
	LiDAR – Angular Resol.	1°
	LiDAR – Sampling Rate	1.8kHz
	LiDAR – Pose	(-0.064,0,0.122,0,0,0)
	Camera (RGB) – Resolution	3280 x 2464 pixel
	Camera (RGB) – Pose	(0.073,-0.011,0.084, 0,0,0)
	Camera (RGB) – Optical Frame	(0,0,0,-1.57,0,-1.57)
Manipulation Aspects	Mechanism(Cartesian/SCARA /cylindrical/parallel/multi-joint/ collaborative/other)	-
	Degree of Freedom	
	DH parameter	
	Max. payload	
Pose: (x,y,z, r,p,y)		

III. INFORMATION MODEL FOR SOFTWARE MODULES

The information model for software modules or Software Information Model, SIM) [17], which can be used in SDR, includes the class shown in Fig. 1 and 2. The SIM is specifically applied only to software modules. The Common Information Model (CIM) [18] can be applied to all types of modules, where modules are classified into hardware modules (e.g. mechanical modules), modules with hardware and software aspects (e.g. controller/sensor/ actuator modules), and software modules. The Common Information Model is similar to Figure 1, but its mandatory classes or attributes differ from those of the SIM [17]. These models provide a standardized framework to ensure the interoperability of software modules across different robotic platforms.

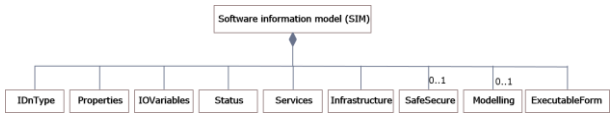

Figure 1. Classes of the software information model (SIM)

The IDnType class has the following attributes: the unique identifier of the module and the software aspects that list software modules operating on a robot. The Properties class provides values for properties that are generally used during the initialization or operation of modules. The properties listed in Table 1 should be defined using the Properties class. The IOVariables class defines the names of variables for data transfer into and/or out of a module, whose typical examples are topic names used as publishers or subscribers in ROS. The

Status class represents the status of a module. The Services class defines the interfaces provided and used by a module within a robot. Its typical examples can be services used in ROS. The Infrastructure class lists communication protocols and middleware that the modules or the robot use. From software aspects, the middleware types can be provided. The SafeSecure class provides the safety-related performance level and security details offered by the module. For software modules, security-related items can be provided. The Modelling class supplies information for the simulation model and/or associated simulation software necessary for executing simulations or digital twins. The ExecutableForm class presents details about the program codes executed to satisfy the module's objectives, whose examples are the program location and how to execute the program and the related libraries.

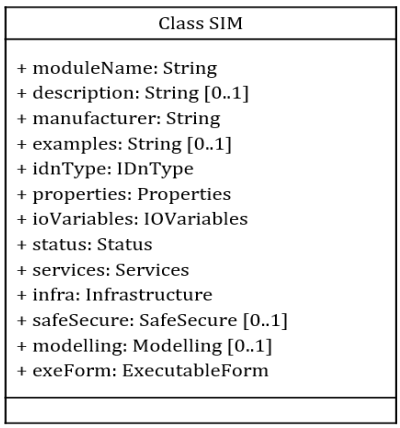

Figure 2. Class diagram of class SIM

IV. SOFTWARE DEFINED ROBOT'S ARCHITECURE BASED ON INFORMATION MODEL

A typical layered structure of SDx is shown in Fig. 3. The structure can be adapted for SDR to manage the increasing variety of software modules efficiently. If managing the software modules is difficult or impossible, management costs will increase, leading to a situation that is not aligned with market demands. To address this problem, the use of standards for software modules is necessary. The more software modules are utilized, the more important these standards become. Middleware and operating systems should be used efficiently in accordance with these standards.

In Fig. 3, the functional module represents a software module that performs functions specific to robots, while the application refers to a application software module created by the user using the interface (or API) provided by the functional module. The challenge arises because the robot's diverse mechanical structures and varying sensor locations restrict the development of various types of modules based on reusing the developed software modules. A crucial tool to resolve this issue is the information model for robots currently developed by ISO TC299 [17,18].

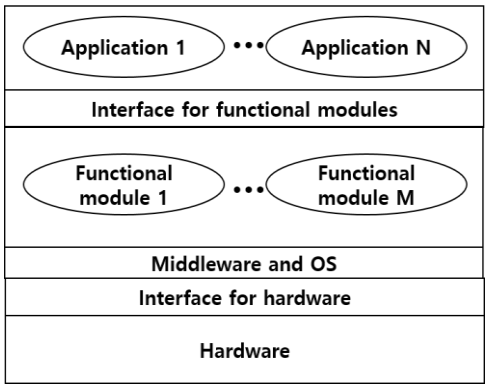

Figure 3. Typical layered structure for SDx.

This paper proposes the structure of Software-Defined Robotics (SDR), as illustrated in Figure 4, based on Figure 3 and the information model. At the initial stage, certain information from the SDR information model is transferred to the information models for functional modules to ensure proper operation.

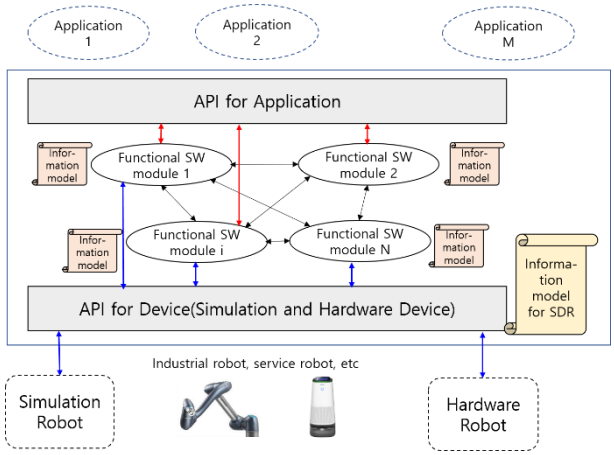

Figure 4. Proposed structure for SDR.

SDR differs from SDV in several ways: SDV is applied to cars and involves the production of millions of units of the same model, whereas robots have diverse applications and may use unique platforms depending on their type. Additionally, the hardware devices used in robots can vary, and the number of robots produced typically ranges from thousands to tens of thousands.

The advantages of this software-defined robot technology are described as follows:

- Ensure flexible expansion, portability, and interoperability of applications.
- Scalable architecture depending on application (can add functional modules with ease)
- Device/system characteristics be controlled by software modules
- Expandable to cloud services (Subscription service, maintenance, modification of usage)

- Accommodating various ideas based on modular structure
- Be able to develop application and function modules separately,

Despite of these advantages, SDR has the following constraints:

- Robot types include industrial, service, medical, and wearable robots, each with its own unique and different mechanical structure.
- The types of sensors used vary based on the robot's application.
- Various types of operating systems and/or middleware are used in robotics.
- As sensors, drivers, and the functional software modules evolve, the interfaces they use may be added or modified.

These advantages and constraints were reorganized based on [21]. Additionally, please refer to [21] for the requirements of SDR.

In Fig. 4, interface-related information for software modules should be clearly categorized into user interfaces, inter-module interfaces, middleware interfaces, and hardware interfaces. Since SDR requires hardware independence, particularly concerning hardware interfaces, the concept of hardware abstraction should be employed. This approach allows software modules to be reused without modification across similar types of hardware. In particular, hardware abstraction interfaces should be capable of interacting with interfaces for both hardware modules and simulators.

V. IMPLEMENTATION AND RESULTS

According to ISO 22166-201, modules are classified into basic modules and composite modules. A basic module is defined as a "module that cannot be decomposed into smaller modules," while a composite module consists of two or more basic modules or other composite modules. From this perspective, SDR can be considered a type of composite module in terms of its information model.

This paper specifically focuses on a mobile platform as an example of SDR, using the TurtleBot3 Waffle Pi, which employs a differential drive for wheel control. The properties listed in Table 1 are essential for controlling and managing the SDR-based mobile platform. Therefore, these properties should be provided to the SDR using the information model defined in ISO/DIS 22166-202, as shown in Table 2.

In ROS, these hardware-related properties are provided in YAML files for use in the launcher, whereas ISO 22166-202 presents them as properties within the information model. Although ROS provides the properties in the launcher file, they should be passed as parameters from the launcher to the corresponding driver. Additionally, some ROS-based robots, such as the TurtleBot, use hard-coded sensor properties rather than specifying them in YAML files, making sensor replacements like LiDAR cumbersome due to the need for source code adjustments. Furthermore, since the installation pose of the sensor is critical, it should be provided as part of the information. For example, because the exact mounting pose of the LiDAR can slightly vary between robots, it should not be hard-coded. Instead, this detail should be included within the information model of either the robot or the specific LiDAR driver module to ensure flexibility and adaptability.

Analyzing these examples reveals two ways to provide hardware information: through a top-level module or a lower-level module with direct access to the hardware. The former approach involves an overhead of passing the hardware information down to the module that actually uses it, since the top-level module itself has no use for this hardware information. A top-level module can be considered a type of composite module, representing the robot as a whole. In ROS, the launcher file is regarded as information used in a top-level module, such as a robot. The latter method aligns with the ISO 22166-201 and ISO/DIS 22166-202 standards, where the software module reads and uses the attributes of the Properties class from the corresponding information model.

TABLE II. ATTRIBUTES IN THE ISO/DIS 22166-202 FORMAT FOR TURTLEBOT

	Values for Turtlebot3 Pi
Properties for control of wheels	<Property name="mobilityMechanism" value = "Differential" type="String" unit="none" description="Wheel Driving method"/> <Property name="wheels_separation" value = "0.287" type="float32" unit="meter" description="separation of two wheels"/> <Property name="wheels_radius" value = "0.033" type="float32" unit="meter" description="radius of two wheels"/> <Property name="max_wheel_acceleration" value="1.0" type ="float32" unit="m/s" description="max wheel acceleration"/> <Property name="max_wheel_torque" value="20.0" type="uint" unit="none" description="max wheel torque of turtlebot3_waffle"/> <Property name="motor_encoder_bits" value = "12" type="short" unit="bits" description="encoder bit length"/>
Names for management of wheels	<Property name="left_jonit_name" value = "wheel_left_joint" type="string" unit="none" descrip-tion="wheel_left_joint"/> <Property name="right_joint_name" value = "wheel_right_joint" type="string" unit="none" description="wheel_roght_joint"/>
Properties for sensors	<Property name="IMU_Pose" value = "0,0,0.068,0,0,0" type="float32" unit="none" description="array, x,y,z,r,p,y"/> <Property name="LiDAR_Pose" value = "-0.064,0,0.122,0,0,0" type="float32" unit="none" description="array, x,y,z,r,p,y"/> <Property name="Camera_Pose" value = "(0.073,-0.011,0.084, 0,0,0" type="float32" unit="none" description="array, x,y,z,r,p,y"/> <Property name="CameraOpticalPose" value = "0,0,0,-1.57,0, -1.57" type="float32" unit="none" description="array, x,y,z,r,p,y"/> <Property name="LiDAR_DistanceRang" value = "0.12, 3.5" type="float32" unit="meter" description="array, min, max"/> <Property name="LiDAR_AngularRange = "360" type="Uint16" unit="degree" description=" Angular Range"/> <Property name="LiDAR_AngularResol" value = "1" type="Uint16" unit="degree" description=" resolution"/> <Property name="LiDAR_SamplingRate" value = "1800" type="Uint32" unit="Hz" description="sampling rate"/>
Simulation model for SDR	<modelling> <simulationModel> <simulator> Gazebo </Simulator> <mdf> <!-- model description file/3Dgraphic file model file --> </mdf> <dynamicSW> <!--provide how to perform the given simulator and the simulation model. --> </dynamicSW> </simulationModel>

	</modelling>

The development tool shown in Fig. 5 is implemented based the Eclipse Remote Application Platform so that it can operate on a cloud basis. Figure 5 shows that the constituent modules for SDRs can be constructed from modules that have already been created and used. It can be known from Table 2 and Fig. 5 that software modules can be reused by modifying the values of attributes used within them. Figure 5 also shows the information editor [19], where basic modules, composite modules, and SDRs are shown in its left small window. The information listed in Table 2 is necessary for operation of TurtleBot. If one of them is not used, the corresponding property should not be included. The software modules have been developed based on ROS 2. A tool has been developed to convert information model to ROS format for ROS developers [20].

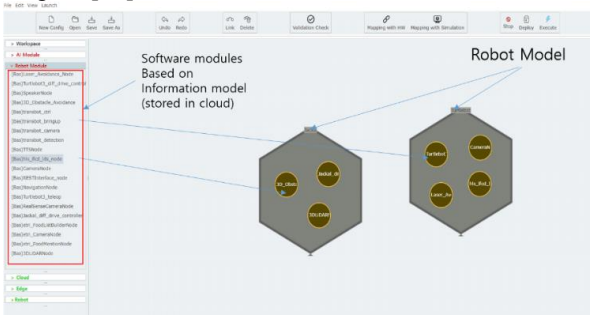

Figure 5. Composition of software modules using developement tool

```
<Module type="SIM">
  <GenInfo>
    <ModuleName>Turtlebot3_WheelDriveModule</ModuleName>
    <Manufactures>https://iic.kangwon.ac.kr</Manufactures>
    <Description>This node enables the internal topics and control for the
turtlebot3</Description>
    <Examples></Examples>
  </GenInfo>
  <IDnType type="Bas">

<mID>0c95eb9da4e156248d865fea1499ca3fff00000000000000ffffffbb420000</
mID>
    <iID>00</iID>
    <informationModelVersion>1.0</informationModelVersion>
  </IDnType>
  <Properties>
    <Property name="mobilityMechanism" value = "Differential" type="String"
unit="none" description="Wheel Driving method"/>

    <OStype type="Ubuntu" bits="64" version="20.04"/>
    <CompilerType>
      <osName>Ubuntu</osName>
      <verRangeOS min="18.04" max="22.04"/>
      <compilerName>CPython</compilerName>
      <verRangeCompiler min="3.8.0" max="3.8.0"/>
      <BitsnCPUarch bits="64" CPUarch="ARM"/>
    </CompilerType>
    <ExecutionType>
      <ExecutionType priorty="1" opType="NONRT" hardRT="FALSE"
timeConstraint="1000" instanceType="Singleton"/>
    </ExecutionType>
  </Properties>
```

Figure 6. XML file of Turtlebot3_WheelDriveModule before depoyment

After the software modules for a SDR (or robot) are composed using the development tool shown in Fig. 5, the software modules and their associated information models are deployed to the SDR or robot for simulation or operation. During this process, the information models of all modules are adjusted to align with the SDR's overall information model (see Table 2), ensuring consistency. Fig. 6 and Fig. 7 present the partial contents of the XML file for the information model before and after deployment, respectively. While Fig. 6 does not match the contents of Table 2, it can be observed that the XML file aligns with Table 2 after deployment. The contents of Fig. 6 and Fig. 7 correspond to the attributes of the Turtlebot3_WheelDriveModule, which utilize the differential drive. Figures 6 and 7 present the properties of the differential drive. Note that the wheel-related mechanical structures can be categorized into differential drive, omnidirectional drive, steering drive, Ackermann steering, and others.

```
<Module type="SIM">
  <GenInfo>
    <ModuleName>Turtlebot3_WheelDriveModule</ModuleName>
    <Manufactures>https://iic.kangwon.ac.kr</Manufactures>
    <Description>This node enables the internal topics and control for the
turtlebot3</Description>
    <Examples></Examples>
  </GenInfo>
  <IDnType type="Bas">
<mID>0c95eb9da4e156248d865fea1499ca3fff00000000000000ffffffbb420000</
mID>
    <iID>00</iID>
    <informationModelVersion>1.0</informationModelVersion>
  </IDnType>
  <Properties>
    <Property name="mobilityMechanism" value = "Differential" type="String"
unit="none" description="Wheel Driving method"/>
    <Property name="max_wheel_acceleration" value= "1.0" type ="float32"
unit="m/s" description="max wheel acceleration"/>
    <Property name="max_wheel_torque" value=" 20.0" type="uint" unit="none"
description="max wheel torque of turtlebot3_waffle"/>
    <Property name="motor_encoder_bits" value = "12" type="short" unit="bits"
description="encoder bit length"/>
    <Property name="wheels separation" value = "0.287" type="float32" unit="m"
description="tutlebot3 wheel separation "/>
    <Property name="wheels radius" type="float32" unit="m"
description="turtlebot3 wheel radius">
      <value>0.033 </value>
    </Property>
    <OStype type="Ubuntu" bits="64" version="20.04"/>
    ....
</Properties>…
```

Figure 7. XML file of Turtlebot3_WheelDriveModule after depoyment

The execution result of the SDR configured in Fig. 5 is shown in Fig. 8. Fig 8 illustrates that software modules developed for SDR can be directly applied to real robots or simulation robot for TurtleBots without modifying the modules' source codes.

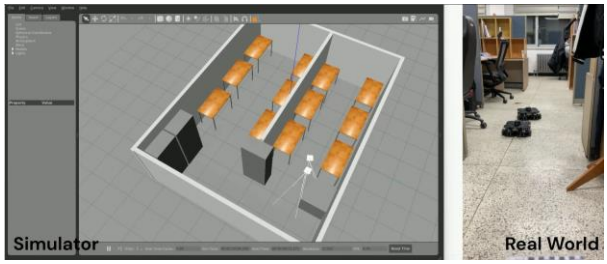

Figure 8. Digital twin robots in Gazebo simulator and the corres-ponding Turtlebots in real world

Figure 9 illustrates the operation of two actual mobile robots (a Jackal robot and a TurtleBot) and one actual mobile

manipulator built using SDR. It demonstrates that both simulation robots and actual robots can be operated by integrating them with robot software modules based on an information model.

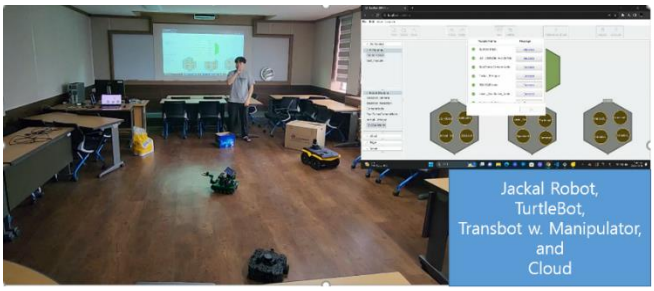

Figure 9. Operation of two mobile robots(Turtlebot, Jackal robot) and one mobile manipulator(Transbot) using SDR

These results demonstrate that the SDRs, configured according to the information models, operate effectively on robot models with various mechanical structures. In other words, it is evident that the information models can be effectively applied to the development of SDR.

VI. Conclusion

This paper has examined the application of Software-Defined x (SDx) principles to robotics, specifically focusing on Software-Defined Robotics (SDR). By decoupling hardware from software, SDR presents a transformative approach that enhances the flexibility, interoperability, and portability of robotic systems. Unlike traditional hardware-dependent configurations, SDR allows for more dynamic development and rapid updates of robotic functionalities, thereby reducing the limitations imposed by physical hardware constraints.

The paper demonstrated the practical implementation of SDR using the TurtleBot3 as a case example. By mapping the actual hardware-related properties of the TurtleBot3 and additional information for the constituent modules necessary for its operation to the standardized information models defined in ISO 22166-201 and DIS 22166-202, the feasibility of SDR was validated through simulations in Gazebo. The result showed that SDR can significantly reduce hardware dependency and increase adaptability, enabling robots to be more easily updated and modified to meet diverse operational needs.

Overall, this paper highlights the significant advantages of SDR based on international standards, including increased flexibility, reduced hardware dependency, and improved scalability. The proposed approach for SDR showed the potential to revolutionize the field of robotics, paving the way for more advanced, efficient, and versatile robotic systems in the future.

Future works should focus on applying the information model standards to a wider range of robotic applications and exploring the potential for further enhancing the scalability and adaptability of SDR-based systems. In addition, further research on hardware abstraction for robots is necessary and application program interface for users.

References

[1] Florian Peston, The Next Step In Digital Transformation Is Software-Defined X. https://www.forbes.com/sites/forbestechcouncil/2021/9/27/the-next-step-in-digital-transformation-is-software-defined-x/?h=439efbb551eb, last accessed 2024/8/1.

[2] -, Software-defined radio. https://en.wikipedia.org/wiki/Software-defined_radio, last accessed 2024/8/1.

[3] IBM, What Is Software-Defined Networking?, https://www.ibm.com/topics/sdn, last accessed 2024/8/1.

[4] -, Software-defined networking. https://en.wikipedia.org/wiki/Software-defined_networking, last accessed 2024/8/1.

[5] Automotive World, What is a software-defined vehicle?, https://www.automotiveworld.com/news-releases/what-is-a-software-defined-vehicle/, last accessed 2024/8/1.

[6] Aptiv, What Is a Software-Defined Vehicle?, https://www.aptiv.com/en/insights/article/what-is-a-software-defined-vehicle, last accessed 2024/8/1.

[7] Marc Amblard, The Software-Defined Vehicle: Enabling and Delivering its Benefits. https://www.linkedin.com/pulse/software-defined-car-enabling-delivering-its-benefits-marc-amblard/, last accessed 2024/8/1.

[8] Vector, One Enabler for the Software-Defined Vehicle. https://www.vector.com/int/en/know-how/software-defined-vehicle-sdv/#c328910, last accessed 2024/8/1.

[9] Bryon Moyer, "Software-Defined Manufacturing. https://www.eejournal.com/article/software-defined-manufacturing/, last accessed 2024/8/1.

[10] Alexander Verl, Managing mass customisation with software-defined manufacturing. https://ifr.org/post/managing-mass-customisation-with-software-defined-manufacturing, last accessed 2024/8/1.

[11] Ali Al-Bayaty, "Software-Defined Robotics - Idea &Approach," arXiv:1708.05935, 2017.

[12] ISO, ISO 8373:2021, Robotics–Vocabulary. ISO, 2021.

[13] OSRF, ROS–Robot Operating System. https:// www.ros.org, last accessed 2024/8/1.

[14] NIAIST, OpenRTM-aist. https://www.openrtm.org/ openrtm/, last accessed 2024/8/1.

[15] S. Han, M. Kim, H.S. Park: Open software platform for robotic services. IEEE Transactions on Automation Science and Engineering, vol.9, no.3, pp.467 - 481, 2012.

[16] OROCOS, Open Robot Control Software. https://orocos.org, last accessed 2024/8/1.

[17] ISO, ISO/DIS 22166-202, Robotics — Modularity for service robots — Part 202: Information model for software modules. ISO, under development, 2024.

[18] ISO, ISO 22166-201, Robotics — Modularity for service robots — Part 201: Common information model for modules. ISO, 2024.

[19] H. S. Park, S.H. Nam, J.H. Park, "Information Model and its Editor for Robotics. Journal of Institute of Control. Robotics and Systems (in Korean)", vol. 29, no.2, pp.135-146, 2023.

[20] D. Yu, H. S. Park, "A Method of Mapping between Information Model and ROS. Journal of Institute of Control, Robotics and Systems (in Korean)," vol. 29, no.10, pp.794-803, 2023.

[21] H.S. Park and et al, "Requirements for Software Defined Robot and its Implementation. Journal of Institute of Control, Robotics and Systems (in Korean)" vol.30, no.5, pp.540-548, 2024.

[22] OSRF, Turtlebot. https://wiki.ros.org/Robots/ TurtleBot, last accessed 2024/8/1.

[23] OSRF, Gazebo tutorials. https://classic.gazebosim.org/tutorials, last accessed on 2024/8/1.