Dealing with configurability in robot systems

Tapio Heikkilä*, Tadeusz Dobrowiecki**, Lars Dalgaard***
*Technical Research Centre of Finland Ltd, Oulu, Finland
**Budapest University of Technology and Economics, Budapest, Hungary
***Danish Technological Institute, Odense, Denmark
* tapio.heikkila@vtt.fi

***Abstract*— In this concept paper the notion of the configurations and configurability is interpreted in the context of robotic and automation systems. A review is given on related methods and techniques, and how configurations relate to system and software life cycles. The modeling principles for configuration based design are outlined and a proposal is made for the design process based on systems and software product line principles, including a completely new way to reuse system model data by storing them as semantic descriptions in SW delivery packages, to be queried by system designers. A tentative design example is presented.**

Keywords— Configuration; Robot Operating System; System and Software Product Line

I. INTRODUCTION

In the near future of the Internet-of-Things (IoT), Cyber-Physical Systems, and Ambient Intelligence, robotic systems will become more versatile and integrated systems of systems, integrating several key technologies. euRobotics aisbl identifies nine key robotic system abilities: configurability, adaptability, dependability, interaction, motion, manipulation, perception and cognitive abilities as well as decisional autonomy [1]. Within varying application constraints, the robotic solutions will realize varying levels of these abilities relying on the adopted combination of the best available technologies.

Started as an open source middleware Robot Operating System (ROS) has become already the mainstream robotic ecosystem. ROS comprises low-level drivers for a wide variety of components, a distributed communication infrastructure, a set of development and visualization tools and a large collection of specific algorithms [2]. Key questions are how to use ROS and the ecosystems in robot system design as old items are swapped for new in the ROS repositories:

- What services and components are available to be included in the application?
- How to match available services and components against the application requirements and select the best?
- How to re-use of earlier designs and how to avoid re-doing specifications?

We answer these questions by expanding our earlier decision support perspective on system design [3, 4] and tackle ROS-based systems as an eco-syst[1]em. Our proposal, developed in the R5-COP project[2], follows principles of Systems and Software Product Line (SSPL) [5], with semantic descriptions made available for the SSPL and application designers.

We discuss next configurations and configurability in robotic and automation systems and present a review on related methods and techniques, and how configurations relate to system and software life cycles. Our main contribution is the proposal for a design concept for reconfigurable robot systems based on SSPL. In addition, we propose to reuse system model data by storing them as semantic descriptions in SW delivery packages, to be queried by system designers. We outline the modeling stages for configuration based design and also present a tentative design example for a ROS based object detection system.

II. CONFIGURATIONS AND CONFIGURABILITY

A. Configuration operations

Configuration and reconfiguration of robot systems deals with setting up and organizing both software and hardware (incl. mechanical) components. Usually all hardware components have software counterparts like driver and data acquisition modules, shifting the modeling toward handling software components and modules. Configuration and reconfiguration operations (examples, see Fig 1) can – and need to – take place in varying degree during or prior to installation, as a result of the end user selection of operating parameters, or even by the autonomous decision of the robotic system itself. Setting and changing the configuration can take place in different phases of the system life cycle. *Static* configuration is set during the development or setup phase of the system and *dynamic* configuration in the run-time.

B. Configurations in robotic and embedded systems

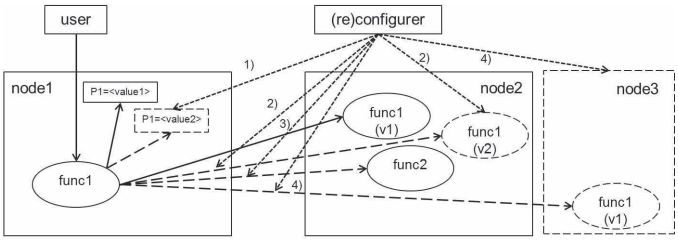
Configurability relates to key architectural properties of system components, e.g. component management and run-time access. ROS by itself does not enforce any explicit component model, leaving space to a non-standardized component structure, in particular regarding configurability. Non-systematic ad-hoc development yields components harder to

978-1-5090-6190-7/16/$31.00 ©2016 IEEE

[2] *Reconfigurable ROS-based Resilient Reasoning Robotic Cooperating Systems* – www.r5-cop.eu

integrate and to reuse. The ROS Industrial Initiative has started to mitigate these problems [6].

Re-usable component models contribute yet to this problem. The Object Management Group (OMG) has specified the CORBA Component Model (CCM), a standard application framework for CORBA components. The Light-weight CCM (Lw-CCM) is targeted towards resource-constrained distributed real-time embedded systems. Currently OMG considers a new standard, Unified Component Model (UCM), to replace Lw-CCM and to be completely independent of any distribution middleware [7]. UCM is expected to meet many needs of modular robot software [8].



1) Change parameter value (set a new parameter value)
2) Update function (upload new SW and renew connection)
3) Use another function (setup a new connection)
4) Use another node (deploy a new node with SW and setup a new connection)

Fig. 1. Configuration operations

Aiming at the integration and the interoperability, OMG has published the Robotic Technology Component (RTC) standard (component model and infrastructure services) applicable to the robotics software development. RTC includes a Platform-Independent Model (PIM) in UML and Platform-Specific Models (PSMs) in OMG IDL. PIM is composed of Lightweight RTC, Execution Semantics and Introspection packages. One of the RTC implementations is the "OpenRTM-aist" [9] which implements also extended safety, according to the IEC61508 Functional Safety standard.

C. Configurations and industrial automation

Configurations are essential also robot applications relying on industrial automation technologies. Automation systems are based on distributed control systems and different fieldbus technology companies have introduced vendor specific tools for handling the variety of HW and SW components, applications and configurations [10]. Interoperability is supported with standards for describing and managing application development, the most important ones are the "function block" standards, IEC 61131-3 (later IEC 61499). IEC 61131-3 is widely adopted and supported in the configuration management tools (tending still to be vendor specific).

The IEC 61131-3 software model ("configuration") specifies the actual resources of a PLC system and provides means to data exchange. Configuration and reconfiguration refer to setting the variable building blocks – function blocks, tasks, programs, resources and access paths. The centralized programmable control of PLCs limits the reuse of the components for reconfiguration [11]. Some SW vendors support the replacement of resources between execution cycles, but such timing cannot be easily controlled [12].

IEC 61499 aims at the challenges of distributed applications, not directly supported in IEC 61131-3. IEC 61499 uses two views of distributed programming: a hierarchy of system, device and resource levels, but also the user-oriented view of applications distributed over the resources. Configuration means setting up the application with the building blocks, i.e. devices, resources and function blocks [13]. It is not yet adopted by the industry and its acceptance in the academic research is said to be questionable [14].

OPC Unified Architecture (OPC UA) is an industrial M2M interoperability communication protocol developed by the OPC Foundation[3] . OPC technologies allow an easy and secure exchange of information between platforms from diverse vendors[4]. Objects and their components are represented in the Address Space as a set of nodes described by attributes and connected by references. Configuration means introducing new or exchanging of existing nodes and setting up the references between them. An optional "Description" attribute describes the capabilities of the node.

D. Domain ontologies

While taxonomies usually provide only a vocabulary and a single relationship between terms (usually a parent/child type of relationship), ontologies provide a much richer set of relationships and allow for constraints and rules to govern those relationships [15]. Regarding configurations, ontologies can be a source for domain languages to describe application requirements and properties of the configurable elements. They can also be used to reason about incomplete requirements and properties.

The IEEE Robotics and Automation Society has developed the standard Ontology for Robotics and Automation (IEEE ORA). Its Core Ontology for Robotics and Automation (CORA) aims at clear definitions of the common (robot) concepts that will permeate all subsequently derived ontologies. CORA relies on Suggested Upper Merged Ontology (SUMO), for search and communication on the semantic web [16]. The standard also includes other complementary ontologies, i.e. CORAX (concepts and relations commonly found in R&A subdomains), RPARTS (concepts useful to represent robot parts) and POS (general notions about position and orientation). Comparable approaches are the Sensor Model Language (SensorML) and Automation Markup Language (AML). SensorML defines processes and components associated with the measurement and post-measurement transformation of observations to enable interoperability. AML defines an XML based format for data exchange between engineering tools. It allows the modeling of physical and logical plant components as data objects encapsulating different aspects and properties, like hierarchy, geometry, kinematics, and logic [17-18].

E. Life cycle and Systems and Software Product Lines

Our approach to handle the reuse and the variabilities within configurations follows the SSPL approach. There are four different aspects to consider [5]:

[3] https://opcfoundation.org/, http://www.commsvr.com/UAModelDesigner/

[4] https://opcfoundation.org/

- Application engineering: from application assets to member products by reusing domain assets within the domain architecture;
- Binding: resolving a variety of optional and alternative features provided by domain and application assets and represented by variability models;
- Configuration management: applications, domain assets, platform releases and products all have versions; products can exist in multiple configurations at any given time;
- Domain architecture: a blueprint for designing the architecture and texture for products.

Product line variability describes the variation among the applications of a software product line in terms of offered features or fulfilled functional and quality requirements [19]. There are two ways to document product line variability: integrated and orthogonal. In the integrated documentation, dedicated concepts are introduced into existing modeling languages or document templates, e.g. variability specific model classes or UML/SysML stereotypes. In the orthogonal documentation, the variability is separated from that of the software development artifacts. Integrated variability modeling increases the complexity. Product line variability is defined redundantly in different requirements and component models, and understanding and tracing it between different models becomes difficult. Orthogonal variability modeling avoids it as only the variability of a product line is defined [19].

Orthogonal variability models describe feature variations in graphical notations originated from Feature–Oriented Domain Analysis (FODA) [20] using *Variation Point* (VP): what varies; *Variant* (V): how varies, *Variability Dependency*: connecting Variation Point and Variant; *Variability Constraint*: the impact of making a decision on Variable Elements; *Artifact Dependency* – connecting Variants with Base Model Elements, and *Decision Set* – the decisions made on the Variation Points. The relationships between parent and child features, i.e. the Variation point and Variants can be [21] *Mandatory*: the child is included; *Optional*: the child is optional; *Alternative*: only one feature of the children (a true XOR) can be selected; and *Or*: one or more children can be included.

Mapping the orthogonal variability model to design assets is restricted. [22] used FODA as a variability and UML as an asset design language, exploiting composition and inheritance. Optional and alternative features (variants) are described as child classes of abstract or concrete classes (variation points). In [23] a UML profile for modeling product line was proposed. Here structural aspects are considered mainly as class diagrams. Variations are expressed with stereotypes <<variationPoint>>, <<variant>> and <<requires>>.

III. Modeling and models for configurations

Our approach to configuration- and configurability modeling builds on the earlier decision support approach [3, 4] and tackles especially ROS-based systems as an ecosystem with the SSPL approach. There are three development tasks involved: for the components, for the Product Line, and for the applications (Fig 2).

1 Modelling the Product Line (PL)

We consider two models for the PL, a variability model, and an architecture model. In our case, the variability model is implemented as a SysML requirements model, with variability annotations. This resembles the FODA approach and that of [23]. The concepts in the requirements models have domain specific annotations using a domain language. The architecture model is described as a UML component diagram, and the requirements concepts expressing variants have links to comparable entities in the architecture model, like to components or interfaces.

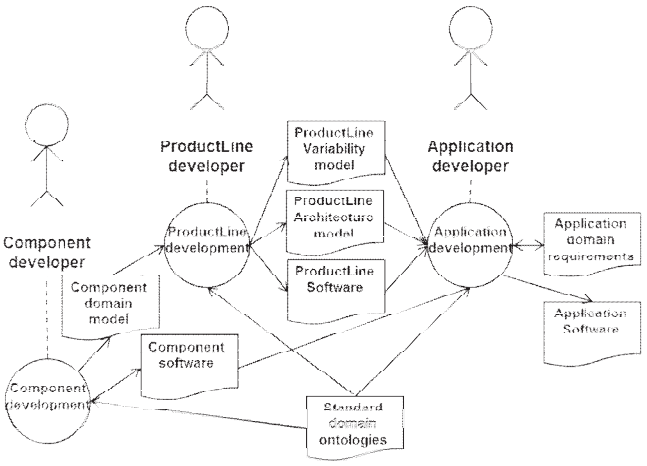


Fig. 2. Design flows between component development, product line development and application (configuration) development.

2 Modelling the components

The components are described with SysML requirements and architecture models. The requirements model presents concepts annotated in the domain language with domain specific annotations. For components, more detailed parametric requirements may be given, e.g. concerning accuracies and execution times for sensors. Optionally, variabilities can be included in the components as well.

3 Modelling the application

The application will also have SysML requirements and architecture models. Requirements are expressed in the same way as the components and the product line with the same domain language. From the SSPL point of view, the application architecture model represents a resolution model, which is derived by matching the application requirements against the SSPL variability model and the parametric constraints originating from the component requirements models.

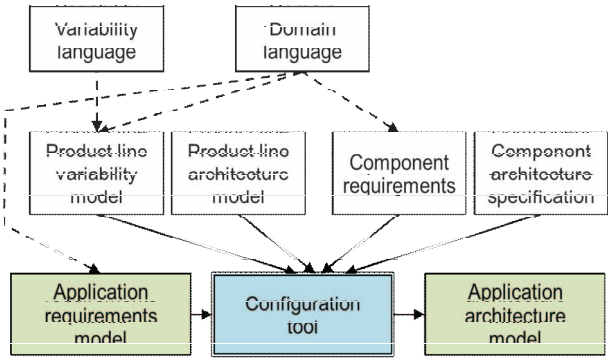


Fig. 3. Knowledge flow and integration in a configuration design tool.

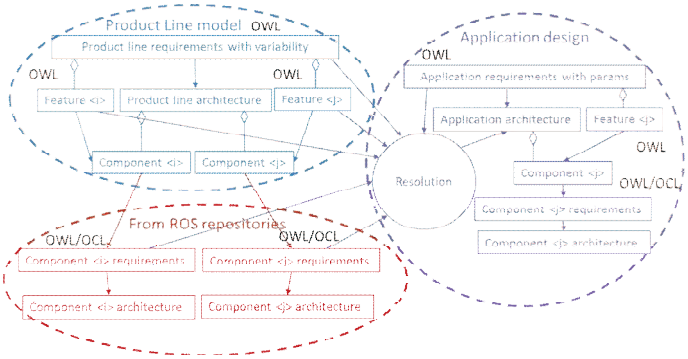


Fig. 4. Configuration design flow.

The domain language plays an important role in the development. It acts as a knowledge description language, shared between the component, product line and application requirements modelling, providing the knowledge for a configuration design tool/environment (Fig 3). The domain language can be any accepted language, but the best would be to rely on well-known standard description languages, like IEEE ORA, SensorML or AutomationML, but currently these seem to lack the versatility needed in robotic applications.

For the robotic application designer, the key issue is what services and components are available. For configuration, one has to match these against the requirements of the application. However, to follow what is newly published in ROS and what becomes quickly obsolete, could be tedious. ROS works as an open platform and an enabler for ecosystems, like Android, iOS or Windows Phone, for smart phone platforms. Such organized "market places" are not (yet) available for ROS and robotics in general. One can query at best in downloaded distribution packages or do auto-search for packages with Linux packet manager command-line tools, browsing ROS documentation or ROS command-line tools, like "`rospack find`".

We propose to use "data sheets" with semantic descriptions – e.g. OWL – of the capabilities of the components and packages based on OWL exports from the requirements parts of the product line and its entities (Fig 4). These descriptions are to be included in the manifest files of distribution packages, which are good for static binding, or stored further in the running components as meta data, for self-adaptive dynamic binding. In addition, to support the tool interoperability, OWL exports from the requirements parts of the product line and its entities have been marked. These local ontologies are needed in a design tool solving the configuration problem.

In configuration management, the key step is the "Component selection, adaptation and deployment". Most of the requirements are given as robot "skills" and modeled as patterns. The decision-making resolves the resources to implement skills according to the requirements and follows the principles of pattern based design [24]. The design flow starts at the highest level of features in the requirements, and resolves the decompositions as given by the product line variability model, matching domain attributes of the capabilities of the product line (components) and the required features. When all the requirement attributes are satisfied, the process continues top-down to the lower levels.

IV. Object detection product line

We have implemented part of our concept by a set of models for designing a configurable object detection system. The models are introduced shortly below.

A. Domain and variability languages

The domain language describing robotic manufacturing applications was defined as a UML sub-profile. The domain concepts – expressing requirements for an object detection and localizing application as well as the capabilities of the comparable product line – included *MotionProvider, Area, SearchArea, Locate, Object, Pose, Location, Point, X, Y, Z, Orientation, rot-X, rot-Y, rot-Z, Measurement, PointCloud, Resolution* and *Accuracy*. Variability of the product line capabilities was described with a variability sub-profile, with prototypes including *VariationPoint, Variant, Variation_XOR, Variation_AND, Variation_OR and Variation_OPT*.

B. Product line model

The product line architecture model for a configurable object detecting and locating system is provided by a UML component model, see Fig 5. The system relies on different sensor technologies to acquire 3D point clouds (3D scanner sensor, profilers with robots, transfer axis and conveyor belts), from which objects are detected, recognized and located, to be further passed to a control system to guide the robot operations. The variable components are integrated here with ROS or sensor specific interfaces, like 2D point measurement profiles. The configuration means selecting a point cloud source and related sensors and devices, depending on the attributes it can provide. For instance, 3D scanners are fast, but triangulation based profilers with motion axis are much more accurate. This

architecture model has been created by gradual expansion of capabilities by introducing new sensors and devices and their compound usage to deliver point clouds.

The product line capabilities are defined as a set of features and their relationships in a SysML requirements model. Fig 6 illustrates the capabilities of the object detecting and locating system product line. All the features are extended with the domain concept stereotypes provided by the domain language sub-profile. It is essential to link the technical specification to the requirements, i.e. to map the required features (blue modules in Fig 6) to product line component references (yellow modules in Fig 6). Further on are the "capability" descriptions for the components as component requirements models, providing information to be matched against the parametric application requirements.

C. Application requirements and application instance

For finding a configuration for the object detecting and locating system instance, application requirements were specified with domain concepts, including: *TransferFloorSearchArea*, *Object*, *Size*, *Location*, *Point*, *Locating*, *Accuracy*, *Resolution*, *X*, *Y*, *Z*, *Max*, *Min*, *Time*, and *Middleware*.

The resolution model – i.e. the application satisfying the requirements – was decided by the designer by matching the requirements against the capabilities of the product line. In this case a tentative example system was created for object detecting and locating (Fig 7). The resolution was composed from a scanner attached to a robot, capable for scanning a wide enough area in the work space.

CONCLUSIONS

We have described how to model configurations in robotic applications following an SSPL approach and showed how to do the modeling with SysML/UML and introduced the idea to store the models as semantic descriptions in SW packages. As an illustration, a solution to an exemplary "configurable object detecting and locating" product line for manufacturing application was presented.

Although the main contribution reported in this paper is the SSPL based design of reconfigurable robot systems, the mentioned models are actually being implemented in OWL and the automated model transformation and configuration computing *Configurer Tool* (see Fig 3), is under development. Drawing on model knowledge base (white modules) the *Configurer* (blue module) will provide services to the robotic system designer to compute component configurations satisfying the *Application Requirements* (green modules) with options to tune the requirements in the light of the obtainable configurations.

In the future models will be extended with various standard representation languages like IEEE ORA. We will expand our model framework towards a knowledge-based tool set accessing the delivery (ROS) packages with semantic component/package descriptions.

ACKNOWLEDGMENTS

This work represents results from the R5-COP project and has been financially supported by the ARTEMIS/ECSEL Joint Undertaking, Tekes – the Finnish Funding Agency for Technology and Innovation – and the Innovation Fund Denmark.

REFERENCES

[1] SPARC – The Partnership for Robotics in Europe. Robotics 2020 Multi-Annual Roadmap, For Robotics in Europe, Horizon 2020 Call ICT-2016 (ICT-25 and ICT-26). Release B 03/12/2015.

[2] Curran W., Thornton T., Arvey B., Smart W.D. Evaluating Impact in the ROS Ecosystem. IEEE Int. Conf. on Robotics and Automation 2015, ICRA 2015, Washington, USA.

[3] Dalgaard L., Heikkilä T., Koskinen J. The R3-COP Decision Support Framework for Autonomous Robotic System Design. 45th Int. Symp. on Robotics (ISR 2014), Munich, Germany, June 2014. pp 278 - 285.

[4] T. Heikkilä, L. Dalgaard, and J. Koskinen, Designing Autonomous Robot Systems – Evaluation of the R3-COP Decision Support System Approach, Proc. of the ERCIM/EWICS Workshop on Dependable Embedded and Cyberphysical Systems (DECS'13)]

[5] ISO-IEC 26550 Software and systems engineering – Reference model for product line engineering and management. First edition 2013-09-01. 34 p.

[6] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, D. Brugali. The BRICS component model: a model-based development paradigm for complex robotics software systems, Proc. of the 28th Annual ACM Symp. on Applied Comp., pp. 1758-1764, New York, USA.

[7] S. Pradha et. al. Towards a Product Line of Heterogeneous Distributed Applications. Tech Rep ISIS-15-117, 4-20-2015, Vanderbilt University, 2015.

[8] G. Biggs, Software modularity: OMG perspective, Introduction to the OMG standards for robotics. Europ. Robotics Forum 2015, Vienna, 2015.

[9] Ando, N., Suehiro, T., Kotoku, T.: A software platform for component based rt-system development: Openrtm-aist. Proc. of the 1st Int. Conf. on Simulation, Modeling, and Programming for Autonomous Robots SIMPAR '08, pp. 87–98. Springer-Verlag, Berlin, Heidelberg (2008).

[10] Pantoni E., A. Mossin, O. S. Donaires, D. Brandao, Configuration management for Fieldbus Automation Systems, IEEE Int. Symp. on Industrial Electronics, ISIE 2007, 4-7 June 2007, Vigo, pp. 1844–1848.

[11] V. Vyatkin, Software Engineering in Industrial Automation: State-of-the-Art Review. IEEE Trans. on Industrial Informatics, Vol 9, No 3, Aug 2013.

[12] R.W. Brennan, P. Vrba, P. Tichy, A. Zoitl, C. Sunder, T. Strasser, V. Marik, Developments in dynamic and intelligent reconfiguration of industrial automation. Computers in Industry 59 (2008), pp. 533–547.

[13] K.-H. John, M. Tiegelkamp, IEC61131-3: Programming Industrial Automation Systems. Springer-Verlag, Heidelberg 2001.

[14] K. Thramboulidis, IEC 61499 Function Block Model: Facts and Fallacies. IEEE Industrial Electronics Magazine, Jan 2010.

[15] C. Schlenoff, E. Prestes, R. Madhavan, P. Goncalves, H. Li, S. Balakirsky, T. Kramer and E. Miguelanez, An IEEE Standard Ontology for Robotics and Automation. 2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Oct 7-12, 2012. Vilamoura, Algarve, Portugal.

[16] A. Pease, I. Niles, J. Li. The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications. AAAI Technical Report WS-02-11.

[17] N. Schmidt, A. Lüder, AutomationML in a Nutshell, AutomationML e.V. Office, Nov 2015.

[18] AutomationML Whitepaper Part 3 – Geometry and Kinematics, AutomationML consortium, Version 2.0, Aug 2015.

[19] Metzger A., Pohl K., Software Product Line Engineering and Variability Management: Achievements and Challenges. FOSE '14, May 31 - June 07 2014, Hyderabad, India.

[20] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Tech Rep CMU/SEI-90-TR-21, Software Eng. Institute, Carnegie Mellon University, Nov 1990.

[21] D. Benavides, S. Segura and A. Ruiz-Cortes, Automated Analysis of Feature Models 20 Years Later: A Literature Review. Information Systems, vol. 35, no. 6, pp. 615 – 636, 2010.

[22] Papakonstaninou N., System design and risk assessment for safety critical control software product lines. Aalto University, 2013. PhD thesis.

[23] Clauss M., Generic Modeling using UML extensions for variability. OOPSLA workshop, Tampa Bay, California, 2001. 8 p.

[24] U. Ryssel, H. Dibowski, K. Kabitzsch. Generation of Function Block Based Designs using Semantic Web Technologies. 14th IEEE Int. Conf. on Emerging Technologies & Factory Automation ETFA'09, pp. 698-705, 2009

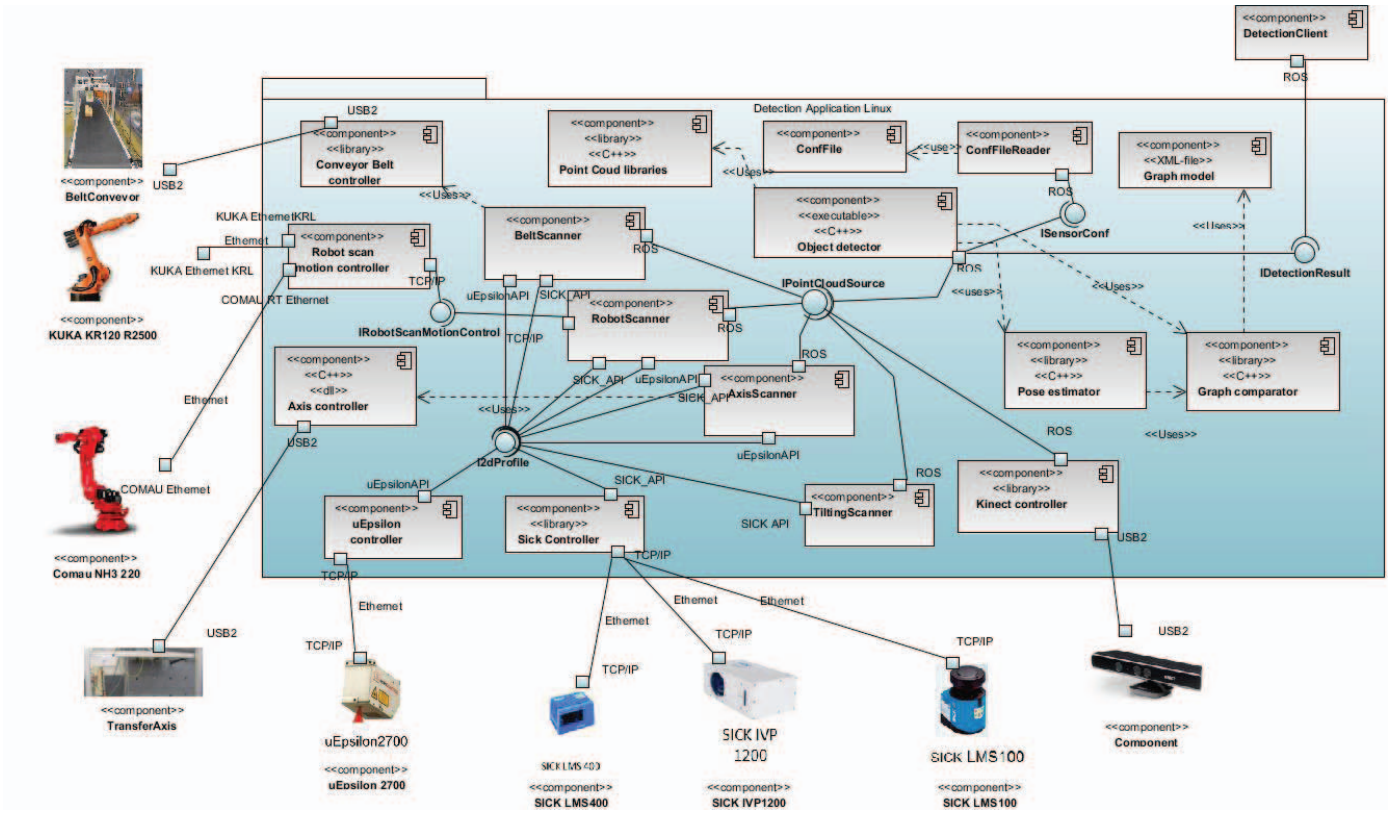


Fig. 5. An object detection product line – architecture model.

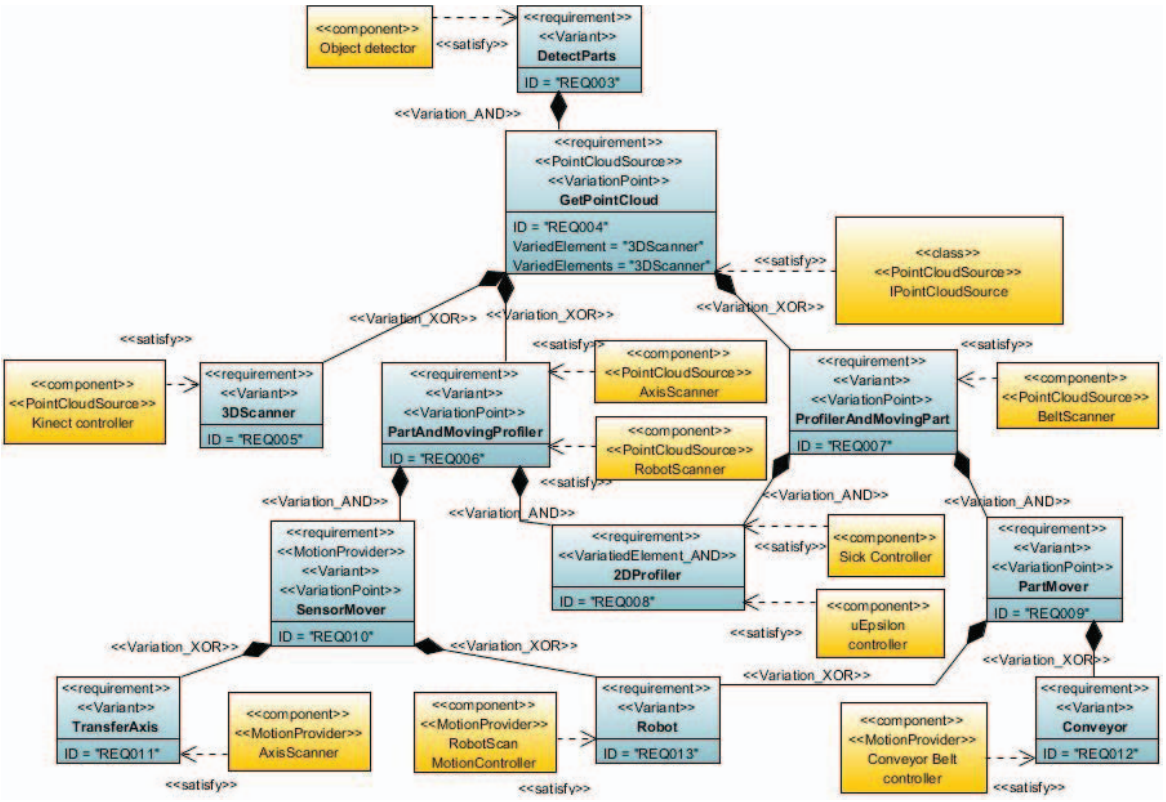


Fig. 6. An object detection product line – variability model.

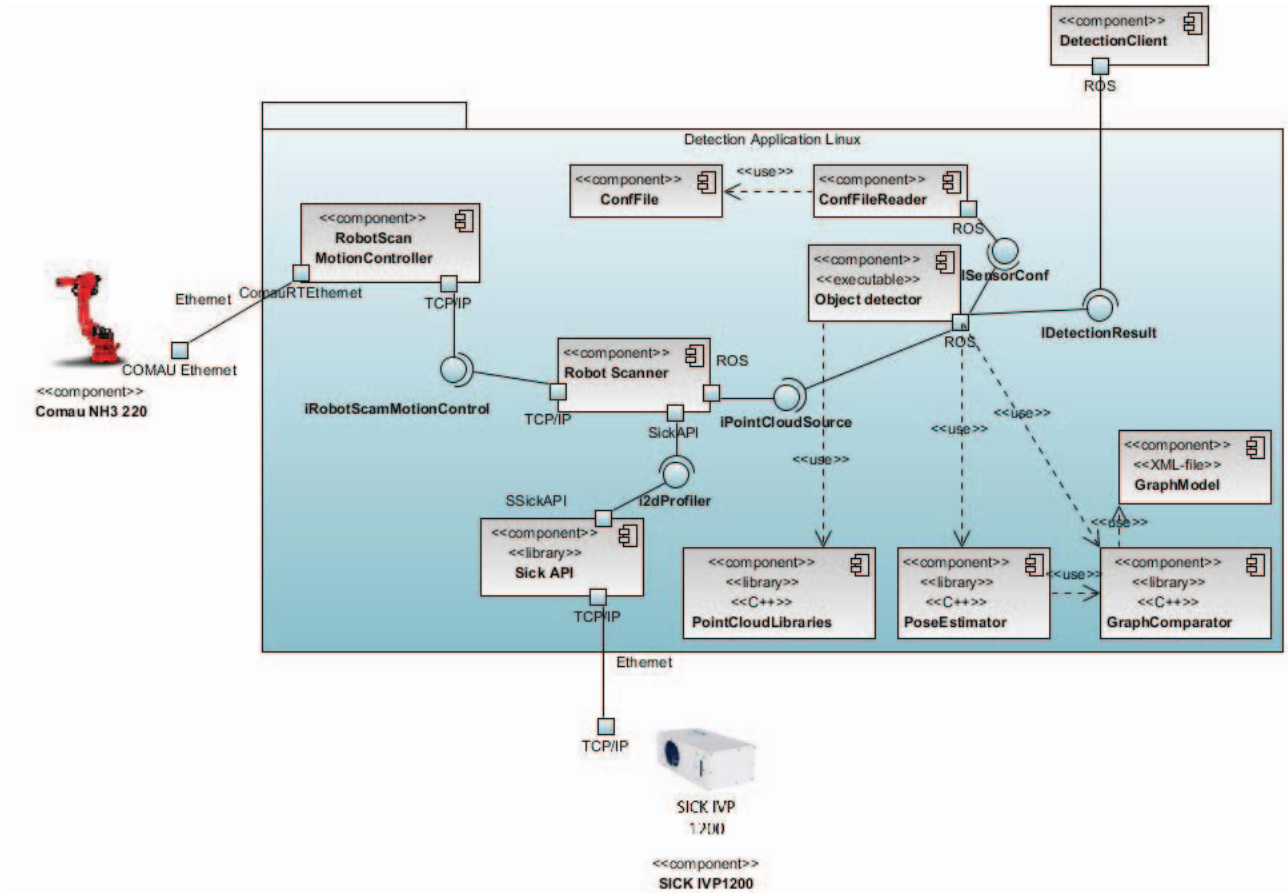


Fig. 7. An object detection system - Application architecture model, derived from the product line model architecture.