Modeling traceability, change information, and synthesis in autonomous system design using symmetric delta lenses

Angeline Aguinaldo[1] and William Regli[1]

***Abstract*— This extended abstract discusses the problem of poor reusability in autonomous system design. We claim that formal semantics for traceability, change information, and synthesis in autonomous system design are necessary to address this problem. Symmetric delta lenses are the proposed construction for modeling these concepts simultaneously. Some initial suggestions for categories that represent knowledge, tasks, and control are provided.**

I. INTRODUCTION

Autonomous systems are employed in millions of factories around the world using computer programs that are specifically tailored to the robotic platform, work cell, material, and task. To add to these degrees of freedom, each of these programs are written by humans that bring their own variations to the programmed solution. One can blur their eyes and see that there exists shared functional behavior between all of these solutions; however, most robotic programs are wholly re-written, incurring significant costs, because there are no standardized ways of synthesizing implementations from functional requirements. This begs for architectures that enable reusability in robotics to minimize cost [1].

Reusability is defined as the ability to minimally change existing programs to adapt to a change in requirements [2]. We define cost as the magnitude of change in a program implementation that achieves the desired change in requirements. A conceptual depiction of this relationship can be seen in Figure 1. These ideas are related to the systems engineering concepts of traceability, change information, and synthesis [3]. Traceability is the ability to identify relationships between artifacts generated throughout the engineering design process. Change information captures how artifacts have been altered according to some data model. Synthesis is the process by which functional architectures and their requirements are translated into implementations.

The components that make up an autonomous system architecture are (i) its knowledge about the world, its actions, and its task goal, (ii) its task and motion plans, and (iii) its control program. In our model, we interpret (i) to be the requirements of the autonomous system, and (ii) and (iii) to be the implementation.

The goal of this work is to formalize that relationship so requirements can be traced to their implementation, change between implementations with respect to requirements can be described, and new robot programs can be formally synthesized. We suggest that a formal framework that captures the above should have the following features:

[1]Computer Science, University of Maryland, College Park

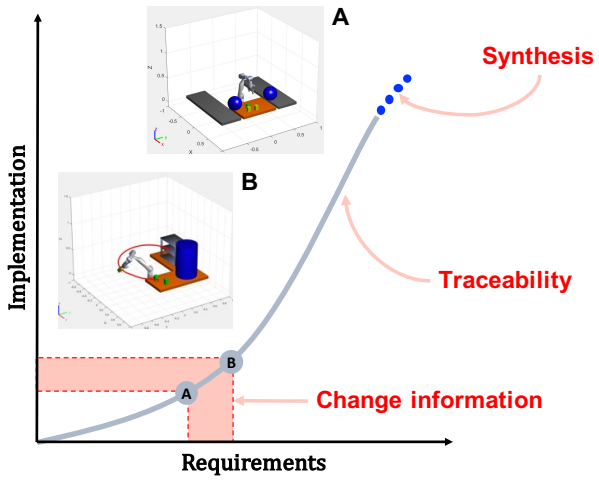


Fig. 1. Conceptual depiction of the relationship between existing implementations and requirements, drawn by the gray curve. **A** and **B** represent scenarios where each system achieves the task of moving yellow and green objects while avoiding blue obstacles, but with different requirements and implementations [4], [5]. Implementations that were automatically synthesized from changes in requirements are depicted using blue circles.

(a) The ability to encode both procedural (task, motion, and control sequences) and declarative (world model) data.
(b) The ability to track varying number of abstraction levels within knowledge, plans, and control.
(c) The ability to encode composite (parts of a whole, decomposition, traceability) relationships and composition (merging, gluing, planning) relationships.
(d) The ability to encode binary relations such as equivalence and similarity with order.
(e) The ability to adhere to constraints demanded by the internal semantics of tasks, knowledge, and control.

II. RELATED WORK

There have been many attempts that tackle (a) and (b) by defining high-level frameworks such as [6], [7], [8], [9]; however, these frameworks do not adhere to consistent semantics making it difficult to achieve (c), (d), and (e). Model-based systems engineering (MBSE) provides concepts for separating functional and implementation details, such as platform independent model (PIM) and platform specific model (PSM) [10], that transcend specific frameworks making them more ready to achieve some of (a) – (d).

MBSE, however, is still limited when addressing all of (c) - (e), namely, composition, similarity, and constraints

imposed by internal semantics. To gain the benefits of MBSE in robotics with the addition of denotational semantics, we can leverage the mathematical concepts from category theory.

There have been many successes in using category theory for MBSE [12], [13], [14], [15], [16], [11], [17], [18]. To our knowledge, there is little work [19] done to leverage categorical semantics to synthesize robotic system implementations. The most relevant work does not discuss how to relate knowledge to the synthesis of robot programs from task plans.

III. Autonomous System Design Components

The following concepts have been well-studied independently but managing within and across these abstraction levels is still an open area of research.

Knowledge. Knowledge about the world, the agent, and its available actions are encoded in model files that adhere to predefined schemas. Some widely accepted formats include the Universal Robot Description Format (URDF) [20] and the Simulation Description Format (SDF) [21]. There has also been work done to describe general information models for robotic knowledge in the form of ontologies [22]. In particular, these ontologies can store kinematic information, world information such as what objects are present and where they are located, the symbolic actions the robot can accomplish, and the action goals.

Plans. Task and motion plans describe how the robot will achieve a goal by identifying a sequence of operations that symbolically update the state of the world [23]. Plans depend on knowledge about the agent and the world to make decisions about what actions are needed to achieve a goal and what kinematic configurations reach each intermediate state.

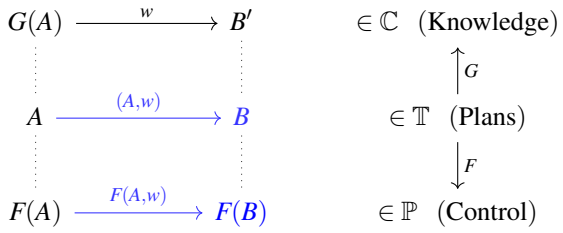
Control. Control refers to the instructions given to the embodied agent that tell it how to actuate. It is possible to generate these instructions independent of explicit knowledge and a plan, and instead based on a conceptual idea a skilled programmer has. In practice, this is often how these robots are programmed. In research, synthesizing control are done by writing in a high-level application programming interface (API) that translates into the target robot language [24], [25], [26], [27], [28], [29], [30], [31].

IV. Categorical Data Model

There exists an intuitive relationship between knowledge, plans, and controls. We propose using symmetric delta lenses to formalize the relationships between these concepts.

Symmetric delta lens are spans in the category of small categories, **Cat**, whose left leg is a discrete opfibration functor, G, and whose right leg is an arbitrary functor, F [33]. Objects within each small category can be thought of as models. Model updates, or deltas, are arrows between models in each category. These arrows are spans [32], where the apex describes the common components and the legs are projections to source and target models [13]. When model updates are executed in the left target category (top) of the lens, the corresponding changes are propagated through the apex to the right target category (bottom) of the lens.

In this framework, traceability can be obtained via the functors, G and F. Change information is captured via the span, or delta, construction. Synthesis of new implementations, namely task plans and control programs, is computed automatically (blue arrows) using the forward and backward propagation operations.



We can begin to populate this high-level framework with categories that would be accessible to the widely accepted data formats and models for each autonomous system component. Some initial suggestions are made in the following sections. Defining functors and checking the consistency of this framework for these categories are left for future work.

A. $\mathbb{C}$ *(Knowledge)*

The category of knowledge configurations, $\mathbb{C}$, is a category whose objects are functors $\mathbb{D} \to$ **Set**, or $\mathbb{D}$-**Inst**, and arrows, w, are spans. $\mathbb{D}$ is an olog category [34] and **Set** is the category of sets. These functors send types in $\mathbb{D}$ to singleton sets in order to instantiate a single knowledge configuration or, in other words, tables with one entry. An arrow is a span in $\mathbb{D}$-**Inst** that describes the knowledge configuration changes between two objects in $\mathbb{D}$-**Inst**. The apex is the shared configuration and its legs are natural transformations that insert and delete data from the apex in the targets.

B. $\mathbb{T}$ *(Plans)*

The category of plans, $\mathbb{T}$, is a category whose objects are monoid categories, and arrows, (A,w), are spans. In each monoid category the object is the set of possible states in the world and the arrows are actions that can symbolically change these states. The apex of a span, (A,w), is a monoid category whose single object is the intersecting set and the legs are functors from the apex to the targets.

C. $\mathbb{P}$ *(Control)*

The category of control programs, $\mathbb{P}$, is a category whose objects are functors, **Grph** $\to \mathbb{M}$, from the category of labelled directed multigraphs to the category of sketches [35] such that sketch constraints are true for all graphs. The arrows between objects are spans where the apex is a functor whose source is the shared subgraph between the two target graphs. The legs of the span are natural transformations that identify the subgraph within the target graphs.

V. Conclusion

We propose an initial framework that models the relationships between knowledge, plans, and control in autonomous systems that captures traceability, change information, and synthesis. This work is ongoing.

REFERENCES

[1] V. J. E. Jiménez and H. Zeiner, "A domain specific language for robot programming in thewood industry a practical example," ICINCO 2017 - Proc. 14th Int. Conf. Informatics Control. Autom. Robot., vol. 2, no. Icinco, pp. 549–555, 2017.

[2] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, and R. da Silva Barreto, "A survey of Model Driven Engineering in robotics," J. Comput. Lang., vol. 62, no. March 2020, p. 101021, 2021.

[3] E. Herzog, "An Approach to Systems Engineering Tool Data Representation and Exchange," Sci. Technol., no. 867, 2004.

[4] Pick-and-Place Workflow Using Stateflow for MATLAB. https://www.mathworks.com/help/robotics/ug/pick-and-place-workflow-using-stateflow.html (accessed: May 8, 2022)

[5] Pick-and-Place Workflow Using RRT Planner and Stateflow for MATLAB. https://www.mathworks.com/help/robotics/ug/pick-and-place-workflow-using-stateflow-and-rrt-planner.html (accessed: May 8, 2022)

[6] A. Scholz, C. Hildebrandt, and A. Fay, "Functional modelling in production engineering workflows," IEEE Int. Conf. Autom. Sci. Eng., vol. 2017-August, no. October 2019, pp. 695–700, 2017.

[7] M. Paczona and H. C. Mayr, "Model-driven mechatronic system development," IEEE Int. Conf. Autom. Sci. Eng., vol. 2019-August, no. August, pp. 1730–1736, 2019.

[8] M. L. Alvarez, I. Sarachaga, A. Burgos, E. Estévez, and M. Marcos, "A Methodological Approach to Model-Driven Design and Development of Automation Systems," IEEE Trans. Autom. Sci. Eng., vol. 15, no. 1, pp. 67–79, 2018.

[9] M. Aragão, P. Moreno, and A. Bernardino, "Middleware interoperability for robotics: A ROS-YARP framework," Front. Robot. AI, vol. 3, no. OCT, pp. 1–10, 2016.

[10] J. Miller and J. Mukerji, "MDA Guide v1.0.1," 2003.

[11] Z. Diskin and T. Maibaum, "Category theory and model-driven engineering: From formal semantics to design patterns and beyond," Electron. Proc. Theor. Comput. Sci. EPTCS, vol. 93, pp. 1–21, 2012.

[12] Z. Diskin, "Algebra of bidirectional model synchronization," Tech. Rep. CSRG573 Dep. Compuitng Sci. Univ. Toronto, vol. 573pdf, no. 7, 2008.

[13] Z. Diskin, Y. Xiong, K. Czarnecki, H. Ehrig, F. Hermann, and F. Orejas, "From state- to delta-based bidirectional model transformations: The symmetric case," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 6981 LNCS, pp. 304–318, 2011.

[14] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter, "A formalisation of constraint-aware model transformations," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 6013 LNCS, pp. 13–28, 2010.

[15] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter, "A formal approach to the specification and transformation of constraints in MDE," J. Log. Algebr. Program., vol. 81, no. 4, pp. 422–457, 2012.

[16] A. Rossini, A. Rutle, Y. Lamo, and U. Wolter, "A formalisation of the copy-modify-merge approach to version control in MDE," J. Log. Algebr. Program., vol. 79, no. 7, pp. 636–658, 2010.

[17] D. Batory, M. Azanza, and J. Saraiva, "The objects and arrows of computational design," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 5301 LNCS, pp. 1–20, 2008.

[18] S. Jurack and G. Taentzer, "Distributed Graph Transformation Concepts," pp. 226–240, 2009.

[19] A. Aguinaldo et al., "RoboCat: A Category Theoretic Framework for Robotic Interoperability Using Goal-Oriented Programming," IEEE Trans. Autom. Sci. Eng., pp. 1–9, 2021.

[20] URDF. https://github.com/ros/urdf (accessed: May 6, 2021)

[21] SDFormat. http://sdformat.org/ (accessed: May 6, 2021)

[22] Tenorth, M., Beetz, M. (2017). Representations for robot knowledge in the KNOWROB framework. Artificial Intelligence, 247, 151–169. https://doi.org/10.1016/j.artint.2015.05.010

[23] M. Mansouri, F. Pecora, and P. Schüller, "Combining Task and Motion Planning: Challenges and Guidelines," Front. Robot. AI, vol. 8, no. May, pp. 1–12, 2021.

[24] M. Racković, "Construction of a translator for robot-programming languages," J. Intell. Robot. Syst. Theory Appl., vol. 15, no. 2, pp. 209–232, 1996.

[25] J. P. Souza, A. Castro, L. Rocha, P. Relvas, and M. F. Silva, "Converting Robot Offline Programs to Native Code Using the AdaptPack

Studio Translators," 19th IEEE Int. Conf. Auton. Robot Syst. Compet. ICARSC 2019, no. Vc, pp. 1–7, 2019.
[26] E. Freund and B. Luedemann-Ravit, "A system to automate the generation of program variants for industrial robot applications," IEEE Int. Conf. Intell. Robot. Syst., vol. 2, no. October, pp. 1856–1861, 2002.
[27] E. Freund, B. Lüdemann-Ravit, O. Stern, and T. Koch, "Creating the architecture of a translator framework for robot programming languages," Proc. - IEEE Int. Conf. Robot. Autom., vol. 1, pp. 187–192, 2001.
[28] B. Kast, S. Albrecht, W. Feiten, and J. Zhang, "Bridging the Gap between Semantics and Control for Industry 4.0 and Autonomous Production," IEEE Int. Conf. Autom. Sci. Eng., vol. 2019-Augus, pp. 780–787, 2019.
[29] D. Cassou, S. Stinckwich, and P. Koch, "Using the DiaSpec Design Language and Compiler to Develop Robotics Systems," Work. Domain-Specific Lang. Model. Robot. Syst., 2011.
[30] J. P. Carvalho de Souza, A. L. Castro, L. F. Rocha, and M. F. Silva, "AdaptPack studio translator: translating offline programming to real palletizing robots," Ind. Rob., vol. 47, no. 5, pp. 713–721, 2020.
[31] M. Bruccoleri, C. D'Onofrio, and U. La Commare, "Off-line programming and simulation for automatic robot control software generation," IEEE Int. Conf. Ind. Informatics, vol. 1, pp. 491–496, 2007.
[32] T. Leinster, Basic Category Theory. Cambridge: Cambridge University Press, 2014.
[33] M. Johnson and R. Rosebrugh, "Symmetric delta lenses and spans of asymmetric delta lenses," J. Object Technol., vol. 16, no. 1, pp. 1–32, 2017.
[34] D. I. Spivak, "Functorial data migration," Inf. Comput., vol. 217, pp. 31–51, 2012.
[35] Z. Diskin, "A Diagrammatic Logic for Object-Oriented Visual Modeling," Electron. Notes Theor. Comput. Sci., no. 4, pp. 1–21, 2006.