2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Experiences with model-centred design methods and tools in safe robotics

Geoffrey Biggs*, Takeshi Sakamoto†, Kiyoshi Fujiwara* and Keiju Anada‡
*Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology, Japan
†Global Assist Co., Ltd., Japan
‡CATS Co., Ltd., Japan

***Abstract*—Development of a system is complex, requiring a well-structured process to manage the range of tasks involved and their work products. There are many models and processes available for structured development, including the well-known Waterfall and Agile models. Recent standards for safety-critical system development utilise the V-model, such as the process given in the ISO 26262 standard for functional safety of road vehicles. However, the process clashes with the commonly-expressed desire for greater reuse of development artifacts in robotics. We have experimented with applying a process, the Object-Oriented Systems Engineering Method, to the design of a robotic wheelchair. This paper describes our application of the process to a safety-critical robot, as well as our use of SysML for managing design information and ZipC for code generation and verification. We discuss our experiences, both good and bad, in order to inform other robot developers of what to consider when choosing a process and tools.**

I. Introduction

The development of a complete system is a complex process. It involves several phases, including design, implementation, verification and operation. A large number of work products are produced that must be tracked for completeness and their relationships to each other. The management of the work and its work products to produce a correct system requires the application of a structured development method, using suitable tools to manage it. This applies as much to robotics as to any other type of system.

There is very little published work in the literature regarding development processes for robotics. Most commonly, a paper makes a brief mention of its "process," and describes something very simple and high-level (for example, [1] and [2]), or describes an architecture designed to support a style of development (such as [3]).

Outside the world of robotics, there are, of course, a large number of development processes available, specified at a range of levels of detail. They range from the traditional Waterfall model of development and the more recent Agile model, to well-regarded processes such as the INCOSE systems engineering process, to any number of ad-hoc methodologies with questionable outcomes. However, it is still more common for a process to be specified at a high level than in detail. The lack of detailed descriptions of steps and work products can make application of a process, and development of a correct system, more difficult.

One of the most detailed and well-structured process specifications currently available can be found in the ISO 26262 standard for functional safety of road vehicles. ISO 26262 describes in detail a development process based on a combination of three V-model processes: one each for system, hardware and software development. The standard provides details on how the steps are to performed, and specifies the work products for each step. (ISO 26262's older brother, IEC 61508 [4], only specifies a very general V-model for development.)

However, the ISO 26262 process is not necessarily ideal for robotics and reflects the static approach found in safety-critical systems standards, where recognition of the need for practices such as iteration and refinement are still not well represented.

1) It does not provide explicit iteration points, which can assist developer decision making.
2) It is based on a develop-once-use-once concept, where reuse of development artifacts, designs in particular, is not encouraged.

Another well-structured process is the Object-Oriented Systems Engineering Method (OOSEM) [5]. This process was designed by the creators of the Systems Modelling Language (SysML) [6] as a way to extend the popular object-oriented approach beyond software, and as a method to structure model-based systems engineering. Like the ISO 26262 process, it describes specific steps and work products.

As part of an investigation into applying structured system development in robotics, we have experimented with applying OOSEM to the development of a robotic wheelchair. This paper describes our experiences in order to inform other robot developers when selecting a development process. This paper does not present an empirical evaluation of the process.

As part of this work, we also report on our experiences using the Systems Modelling Language (SysML) for system design [6], and the ZipC tool for formal state machine specification, verification and code generation [7]. The use of tools to support the development process, providing information management and formality, is now regarded as software engineering best practice, particularly in critical systems development.

OOSEM is well integrated with model-based techniques and SysML. We selected OOSEM because it is oriented towards the model-based development concepts supported by SysML and specifies many work products to be produced in SysML, which we were already using to model the robot's

978-1-4673-6357-0/13/$31.00 ©2013 IEEE

design. Although some other processes make use of models, such as the INCOSE System Engineering process, none have as explicit support for model-based techniques as OOSEM. We also selected it based on its explicit support for re-usability of design artifacts. SysML was being used based on its popularity (there is a wide, and expanding, range of tooling available) and on its ability to represent the entire robot design, not just hardware or just software.

The next section describes the robot system. Following that, section III briefly describes SysML. Section IV describes the OOSEM process, with some examples from our robot design. Section V discusses using ZipC to generate and verify the state machine code. Discussion and comments on the process and tools are given in section VI.

II. Robot wheelchair

The target application for this work is a robot wheelchair. The wheelchair provides safe motion to the rider, who provides the control input directing that motion. "Safe motion" means that the robot avoids collisions whenever possible and minimises impact when a collision is inevitable. The wheelchair must be robust to internal software and hardware failures to achieve this safety.

The robot design is based on partitioning the safety-critical and non-safety-critical parts into separate sub-systems. This technique is often known as a "safety bag" in safety-critical systems design [8], and has been applied in robotics before [9].

The safety-critical part is contained in a black-box system that presents a simple interface for motion control. It is responsible for guaranteeing the safety of the robot by monitoring the area around the robot for obstacles and by detecting contacts with the robot and altering the robot's speed as appropriate to the current safety level. It is also responsible for monitoring the health of the robot's critical software and hardware and stopping the robot in the event of a failure.

The non-safety-critical part uses the safety-critical part as a black-box provider of mobility. It is free from all responsibility for the safety of the robot.

The advantage of this design is it frees the high-level motion control of the robot, such as a path planner, from the constraints of being safety-critical. This allows for the application of less expensive development methods and system components, while still ensuring the specified level of safety in the system as a whole.

This application required a multi-person development team. For example, the hardware was built from scratch, rather than just loading the software into an existing robot body. This required that the development team include both a software engineer and a hardware engineer. We also employed a professional SysML modeller to assist with the modelling process, and an expert in using the ZipC tool for state machine verification.

The robot is shown in Figure 1. The robot design was specified using SysML, described in the next section.

(a) CAD hardware design

(b) Constructed robot

Fig. 1: The safe robot wheelchair

III. Systems Modelling Language

SysML is a profile of the Unified Modeling Language (UML), designed for systems engineering. It has been standardised by the OMG. As well as reducing the emphasis on software found in UML, it also adds additional diagrams.

1) Requirements diagrams for managing requirements, an important part of any systems engineering process.
2) Parametric diagrams for simulation and quantitative system analysis, allowing the integration of engineering analyses into the system model.

SysML's capability to represent the full range of systems engineering tasks, including requirements, behaviour, structure and even verification and design trade-off analyses, makes it well-suited to systems engineering.

As with any visual modelling language, and like its parent, UML, SysML relies heavily on tool support to avoid becoming unwieldy. Tools provide facilities for managing model packages and element databases, for generating implementations from models and for conducting model simulations. Recent tools can even call out to specialised tools, such as Matlab, to perform these simulations. In this work, Enterprise Architect [10] was used as the modelling tool.

Unlike other domains, which have a large body of literature on applying SysML-based approaches, there has been little prior contact between SysML and robotics. Notably, [11] argues that SysML profiles for robotics should be created.

These profiles would ease the use of modelling in robotics and drive its adoption. [12] investigated using SysML to model a manipulation task. [13] used SysML to model a robot application that was then transformed automatically into software components for RT-Middleware (notably, they used a Platform-Independent Model/Platform-Specific Model approach). [14] used SysML and Simulink in combination to model the controller of an inverted pendulum robot, and found the combination to be of benefit. However, none of the above articles have discussed the process guiding the use of SysML.

We applied SysML to specify the robot design. It was used for:

1) specifying the domain in which the robot would operate, including other entities;
2) specifying the use cases (including their scenarios) and requirements;
3) designing the system decomposition;
4) designing the connections between system components, including the type of information flowing between them;
5) specifying tests to perform to verify the design; and
6) managing traceability of the requirements.

Space limitations prevent us from presenting the entire model in this paper. Example diagrams are shown in Figures 2, 3 and 4. Figure 2 shows the activity diagram defining the behaviour of the robot as it moves. Figure 3 gives an internal block diagram showing the abstract design of the robot's implementation. Figure 4 gives an internal block diagram showing the implementation of the Controller from Figure 3.

The creation of the SysML model was managed with OOSEM, described in the next section.

IV. Object-Oriented Systems Engineering Method

The Object-Oriented Systems Engineering Method (OOSEM) is a methodology for the analysis, design, specification and verification of any kind of system. It is based on object-oriented methodology and the Systems Modeling Language (SysML).

OOSEM is capable of specifying any part of the system, including:

- requirements and constraints;
- logical and physical implementations;
- tests;
- processes and usage patterns; and
- people, including users, operators, maintainers and bystanders.

OOSEM uses the top-down process commonly found in object-oriented methodologies. It is based around specifying system requirements and the scenarios for them. Techniques such as model-based abstraction and traces are used to adapt the system as requirements and targets change.

As with all Model-Based Systems Engineering processes, OOSEM relies heavily on a system model, in this case specified using SysML. The model contains all aspects of the system, from requirements through to behaviour descriptions and implementation descriptions.

An important property of the artifacts produced by OOSEM is their re-usability. Following OOSEM ensures that an abstract model of the system, independent of any specific implementation, is available. This is known as the "logical decomposition" or "logical implementation" in OOSEM and more generally as the Platform Independent Model (PIM). It can be adapted for any target environment by repeating the final step in the OOSEM process within the constraints for that target environment. The adapted design is sometimes known as a Platform Specific Model (PSM).

OOSEM is illustrated in Figure 5. Note the separation between black-box system specification, logical system specification and physical implementation specification. For a complete description of the process, we refer the reader to [5].

In applying OOSEM to the wheelchair, we split the model into two phases to follow the logical/physical split.

1) A PIM for a mobile robot base that provides safety.
2) A concrete implementation (PSM) built around the base (the robotic wheelchair).

The robot wheelchair model includes and uses an abstract model of a generic safe mobile base. It defines its own requirements, activities, etc. to complement those from the abstract model, which only specify aspects related directly to safe motion. Splitting the model in this way had two advantages for our design process.

First, the design process was able to concentrate initially on the safety aspects of the robot, which are relatively simple compared with the complexity of the full wheelchair system.

Second, the safe mobile robot base design is re-usable. If, for example, we were to design a dependable robotic transporter for warehouses, we would be able to begin with the general safety aspects of a mobile robot already complete, needing only to add the safety aspects and new requirements of the new application.

V. ZipC

Safety-critical system development standards, such as IEC 61508 and ISO 26262, recommend the use of automated tools for any task that can be automated. The most common example of automation is code generation. Safety standards also recommend tool-based verification of correctness.

ZipC is a tool for embedded system design and implementation. It can be used for specifying, testing, verifying and inspecting Finite State Machines. It is used to specify a model of an FSM, from which C code implementing that FSM is generated. The model can also be simulated to check for correctness. The generated code can be executed and monitored in real-time using the tool. This feature is particularly used for detecting when the FSM enters states it should not given the inputs, which indicates a flaw in the logic. ZipC is also capable of other verifications, such as finding states that cannot be reached, or multiple transitions from a state being triggered at the same time.

The controller of the robot uses an FSM to monitor the robot's level of safety. After initially modelling this FSM in

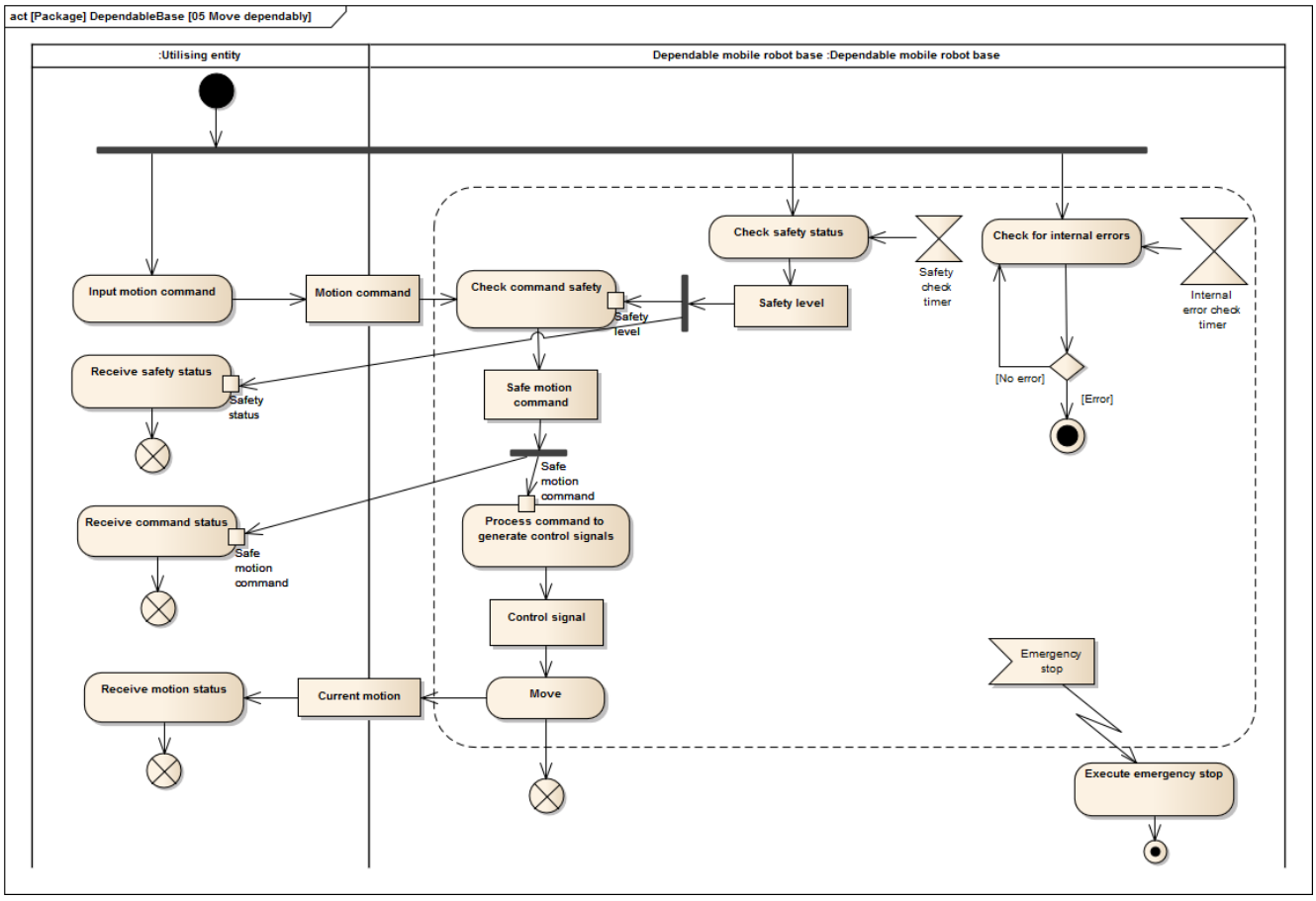

Fig. 2: An activity diagram specifying a use case scenario

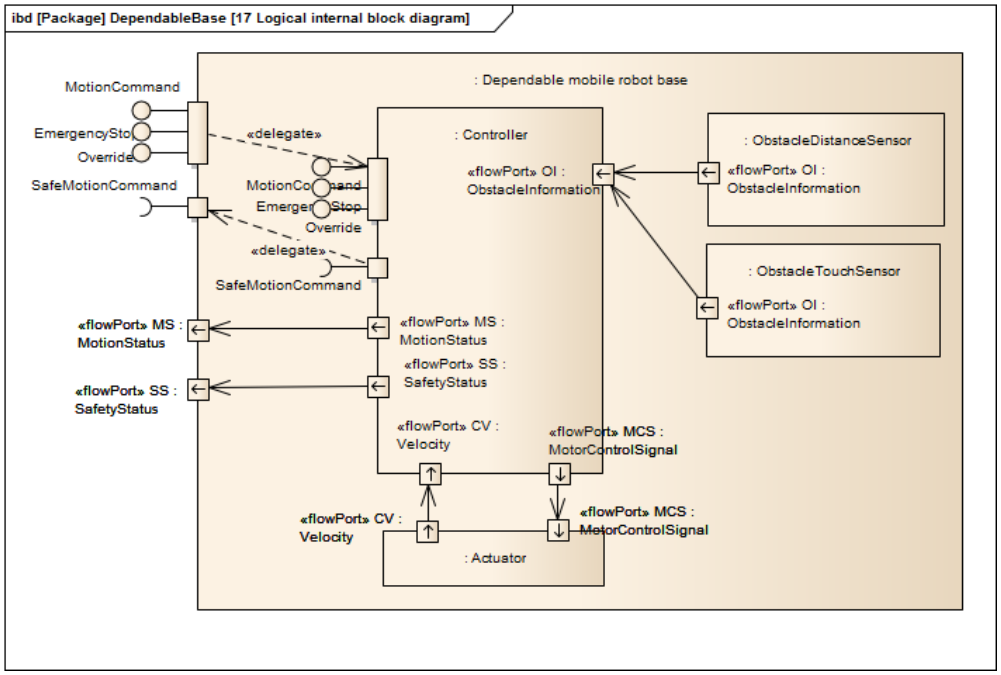

Fig. 3: The internal wiring of an abstract safe mobile robot

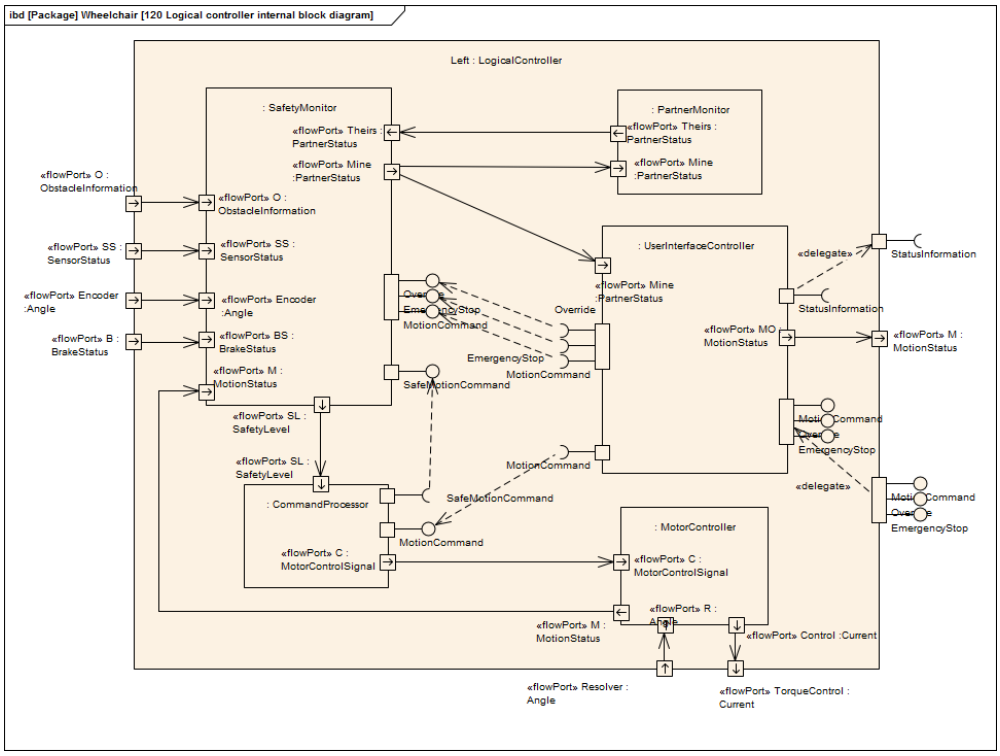

Fig. 4: The internal wiring of the abstract safe mobile robot's controller

SysML as part of the OOSEM process (behaviour specification), we used ZipC for verification and code generation. The FSM model was re-constructed in ZipC.

From the model, a complete state machine implementation was generated in C, which was dropped into the robot controller's source code. The FSM was executed by the component responsible for safety monitoring, with events for the FSM being linked to data received on its ports.

The generated code was executed on the robot hardware with ZipC connected. ZipC recorded the inputs, outputs and states of the FSM over time. It used this data to display which states had been correctly entered, which had been incorrectly entered, and which had been incorrectly not entered. This information is displayed in a matrix format, such as that shown in Figure 7 (Japanese original shown in Figure 6. The important aspect of this diagram is the tabular layout with events down the left side, states across the top, and cells representing the behaviours to perform in each state corresponding to each event (a slash indicates no action for an event in that state). Cells with no colour are behaviours that were not executed during testing. Coloured cells were executed, with red cells being executed the most and green cells executed the least. The number of executions corresponding to each colour are set prior to testing based on the goal of the analysis.

VI. Discussion

In this section, we present our observations on applying OOSEM to the robot wheelchair development, as well as on using SysML/Enterprise Architect and ZipC.

A. Benefits

1) Re-usability of design artifacts: A strength of OOSEM is its allowance for reuse of design artifacts through the explicit creation of a logical architecture, which is then refined into a physical architecture. The structure of OOSEM even allows for multiple levels of reuse by fitting further-refined logical architectures between the initial logical architecture and the final physical architecture.

We were able to apply this method of reuse to the robot wheelchair with no difficulty. The system concept was divided into the core "safe motion" concept and the "robot wheelchair" concept. We found that this allowed us to concentrate on the needs of safe motion in general before thinking about how to apply safe motion in the specific case of a wheelchair, not to mention the large number of other requirements involved in building a robot wheelchair.

This reuse occurred at all levels of the design phase, from requirements and behaviour specification through to implementation specification, and will be immediately recognised by anyone familiar with object-oriented software concepts. Of particular benefit, due to the use of SysML, was that the reuse of both hardware and software designs was possible.

For example, the "safe motion" concept design, part of which is shown in Figure 3, contains a "Controller" element with certain interfaces and properties. When refining the design in the "robot wheelchair" concept, a "LogicalController" element was added (the internal design of which is shown in Figure 4) that inherits from the "Controller" element its design, but also extends and refines it with information

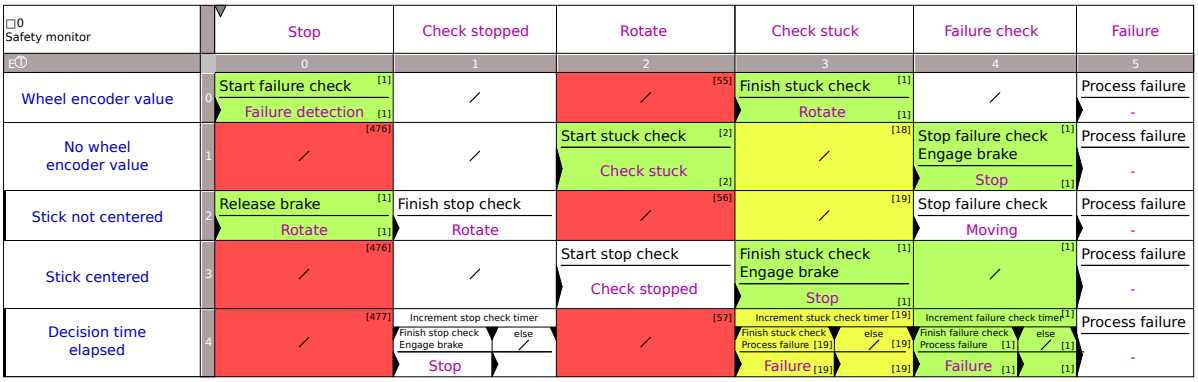

Fig. 7: Verifying the state machine using ZipC

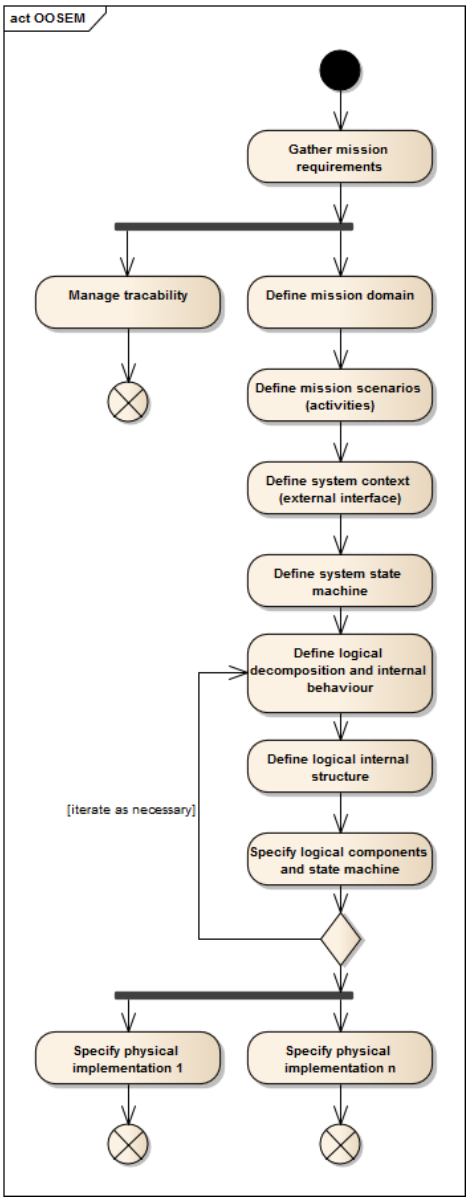

Fig. 5: A very brief overview of the OOSEM process

Fig. 6: Verifying the state machine using ZipC (Japanese original)

specific to the design choices made at that point. For example, the use of torque control rather than the generic "MotorControlSignal" output. In an alternative design, a different method of motor control may be used; such a controller would inherit from the "Controller" element to reuse its properties but refine them in a different way. SysML provides strong support for the concept of refining a design, and OOSEM leverages this.

We believe that this method of re-usability will be very important in future robot development. Rather than starting a design from scratch, a base robot application concept with an existing model can be selected and refined to a complete design by adding the requirements and constraints for the target application. We also believe that, combined with a repository of existing implementation models and software components, the ability to automatically generate implementations of the software parts of a system will be well-supported by OOSEM.

2) Unified high-level hardware/software design: SysML is capable of representing both hardware and software at an abstract level as system components. This allowed us to reason about the "internals" of the controller (i.e. its software) in the same model as its external hardware design. This extended to designing the individual software modules and their interfaces, which were later realised as software components in RT-Middleware.

3) Correspondence of software design to implementation: We used a component-based software architecture to implement the controller software. This turned out to fit well with SysML's specification of design, which is similarly component-based. Although code generation was not avail-

able, the direct correspondence between each SysML component within the controller and the software component to implement it, including the ports connecting them, made creation of the software component network simple.

The similarity of SysML to UML (due to it being a UML profile) made it well-suited for much of the software modelling needs.

On a related note, traceability was simple to ensure using SysML. Its support for linking requirements, implementations and tests together made it easy to track what requirement was implemented by what system components, and how they should be tested.

4) High-level models: We found SysML to be particularly good for abstract design models. For example, it was simple to adjust the design of the system, such as what information two components exchange, in the model during design review meetings, and review the impact of this change on other parts of the system.

The abstractness of the design representation was also an aid to understanding. Although the design team included a hardware engineer and a software engineer, they were able to communicate their design concepts easily. By contrast, the software engineer was not able to read the hardware engineer's circuit diagrams without needing to ask what specific symbols meant. The system model was a benefit to understanding and communication.

5) Code generation: The ZipC state machine model was sufficiently formal to allow automated code generation. This removed a significant potential source of faults from the development process. Manually translating between a model of a state machine and the implementation of that model would have been error prone and could have introduced faults into the system implementation that would not have been detected for some time. This would have raised development costs, especially when an introduced fault led to a failure, which would have required debugging and repair effort and impacted on certification.

The code generation also eliminated the time usually required to implement the state machine.

6) Automated verification: ZipC allowed the state machine design and implementation to be verified more comprehensively than manually-specified tests would have. This gave us greater confidence in the correctness of the robot's design and implementation. We were rewarded for using an automated tool by a controller implementation that had no faults in its state machine after implementation.

B. Difficulties

We encountered several difficulties, which we hope will inform other robot developers in their choice of tooling and development process.

1) Difficulties with OOSEM: Although OOSEM is strong in producing re-usable design artifacts, we did encounter problems when we applied it.

The lack of states beyond design led to an inconsistency in the complete development process. This inconsistency was caused by a mis-match between the final work products of the OOSEM process and the next stages of development. Although to be expected when using a process that does not cover the entire system life cycle, this inconsistency was found to be a greater problem than expected in terms of guiding the development. This lead, in particular, to problems with verification of work products and the design. Any systems engineer looking to apply OOSEM must find additional processes to support the other phases of the system life cycle. In our opinion, this makes it harder to choose OOSEM over a more completely-specified process, such as the ISO 26262 process.

We also found that OOSEM placed most of its emphasis on system structure. For example, while it explicitly provides for specifying the interfaces of a component, it makes no allowance for specifying the timing of information going into and coming out of that interface. Steps should be added to the process to specify timing information, using SysML's timing diagrams.

Finally, the OOSEM process made no allowance for common safety-critical system development activities, such as a hazard analysis. Although this is not a problem for a non-safety-critical robot, it is a significant gap for safety-critical systems.

2) Tool problems: We used Enterprise Architect version 9 as the SysML modelling tool in this project. We found its support for SysML to be incomplete, which required ad-hoc methods to work around and reduced the benefit of using a model to achieve consistent design. Additionally, its model checker did not support SysML, which made it difficult for inexperienced modellers on the development team to know if their model was semantically correct.

We encountered difficulties in using Enteprise Architect and SysML for requirements management. The large number of requirements involved was difficult to manage using the SysML graphical views. Other tools, such as IBM's Rhapsody, solve this by using tabular requirements management and only visually modelling specific requirements for analysis purposes. At the back end, all requirements are still stored in the model so consistency and traceability are ensured. (Requirements management is a well-known problem in systems engineering.)

The lack of code generation from the SysML model was also a problem. Although code generation of the controller's state machine was used, no code generation for the remainder of the software was available. This is a significant gap in the tooling used. We recommend that robot developers choose tools that provide code generation of as much as possible.

Tool integration was a notable concern: Enterprise Architect and ZipC share no common formats. We had to re-model the state machine in ZipC to use its code generation and verification capabilities. This is a potential source of faults that would not exist with better tool integration.

3) High-level models: Although SysML was good for specifying the high-level design model, specifying detailed models was considerable effort. Without some benefit to this effort, such as code generation (whether software source code or hardware description language specifications), it is

difficult to justify the cost of producing a design that is more detailed than necessary for experienced engineers to implement accurately. Such tools do exist; we neglected to recognise all our eventual needs at the start of development, and did not choose a tool that met those needs.

We also believe that, given its use of a component-based construction style, designing a system in SysML will lead directly to generating complete software systems from the modelled components combined with a repository of pre-created software components. This method of system construction is not yet available in the tooling, however.

4) Correspondence of hardware design to implementation: Unlike the software design correspondence, hardware design corresponded less well between the model and actual designs. Hardware can be modelled and designed in tools such as Modelica, CAD (utilised in this project) and electrical circuit design tools (also utilised). However, there was a greater disconnect between the SysML model and the CAD model/-circuit diagrams than there was between the software design and the system model.

5) Determining "completeness": During the modelling process, we found that the hardest part was knowing when to stop. Given enough time, it is possible to model even the smallest detail of a system in SysML; however, this would not be much use to the development process. On the other hand, the model must have sufficient detail to unambiguously describe the system. Knowing this point is, in the view of the authors, dependent on the needs of the system, the capabilities of its implementors (for example, inexperienced software developers require more guidance), and probably something that can only be learned from experience.

We theorise that code generation can ease this problem: when to stop modelling finer and finer details is when a complete system can be generated.

VII. Conclusions

The need to use structured development methods to ensure correct system development is well-known, although rarely applied to robotics. There is little guidance in the literature to assist robot developers in choosing a process or tools. We have experimented with applying a development process with explicit reuse to the development of a robot wheelchair. As part of this work, we have also applied SysML and ZipC, tools for model-based development.

Our experience has shown that OOSEM brings a strong focus on re-usability to the development process. We believe that, when coupled with a repository of hardware and software components, OOSEM will be well-suited to model-based system development and can contribute to reduced development times. We also found that SysML contributes in the same way, due to its focus on component decomposition and interface specification for system design. The ZipC tool turned out to be the most usable, allowing us to model, verify and automatically generate the controller's state machine implementation.

However, we cannot fully endorse the use of OOSEM in robotics. Its focus on system structure ignores aspects such as timing which are very relevant in real-time robot applications. It also only specifies the design phase, meaning a systems engineer must look for additional processes for the other life cycle phases and ensure that those processes fit with OOSEM. We recommend that robot developers seek out a development process that covers the complete system life cycle while providing strong support for model-based design and reuse.

In future work, we plan to experiment with the ISO 26262 process, which is an obvious candidate for developing safe UGVs. In particular, we wish to understand how it can support iterate-and-refine development of robots and the use of model-based design reuse.

References

[1] J. Osada, S. Ohnaka, and M. Sato, "The scenario and design process of childcare robot, papero," in *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, ser. ACE '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1178823.1178917

[2] R. Hanai, H. Saito, Y. Nakabo, K. Fujiwara, T. Ogure, D. Mizuguchi, K. Homma, and K. Ohba, "Proposal of architecture and implementation process for IEC61508 compliant, dependable robot systems," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, 2012, pp. 1218–1223.

[3] G. Hirzinger and B. Bauml, "Agile robot development (ard): A pragmatic approach to robotic software," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Oct., pp. 3741–3748.

[4] "IEC 61508-2 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 2: Requirements for electrical/-electronic/programmable electronic safety-related systems," 2010.

[5] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, 2009, ch. 16.

[6] (2010) OMG Systems Modeling Language (OMG SysML). [Online]. Available: http://www.omg.org/spec/SysML/1.2/

[7] (2013) CATS Co., Ltd. – ZipC. [Online]. Available: http://www.zipc.com/english/guide/products.html

[8] P. Klein, "The safety-bag expert system in the electronic railway interlocking system elektra," *Expert Systems with Applications*, vol. 3, no. 4, pp. 499 – 506, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/095741749190175E

[9] F. Py and F. Ingrand, "Dependable execution control for autonomous robots," in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 2, Sept.-2 Oct., pp. 1136–1141 vol.2.

[10] (2013) Enterprise Architect. [Online]. Available: http://www.sparxsystems.com/products/ea/index.html

[11] H. Bruyninckx, "Robotics software: The future should be open [position]," *Robotics Automation Magazine, IEEE*, vol. 15, no. 1, pp. 9 –11, march 2008.

[12] K. Ohara, K. Iwane, T. Takubo, Y. Mae, and T. Arai, "Component-based robot software design for pick-and-place task described by SysML," in *Ubiquitous Robots and Ambient Intelligence (URAI), 2011 8th International Conference on*, nov. 2011, pp. 124 –127.

[13] M. A. A. Rahman, K. Mayama, T. Takasu, A. Yasuda, and M. Mizukawa, "Model-driven development of intelligent mobile robot using Systems Modeling Language (SysML)," in *Mobile Robots - Control Architectures, Bio-Interfacing, Navigation, Multi Robot Motion Planning and Operator Training*, J. B. edkowski, Ed. InTech, 2011, ch. 1, pp. 21–38.

[14] D. Phaoharuhansa and A. Shimada, "An approach to SysML and Simulink based motion controller design for inverted pendulum robots," in *SICE Annual Conference (SICE), 2011 Proceedings of*, sept. 2011, pp. 2190 –2193.